日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００１年 ３月 ６日

出願番号
Application Number: 特願２００１－０６１５４４

出願人
Applicant(s): 株式会社日立製作所

U.S. Appln Filed 8-29-01
Inventor: M. Kaminaga et al
Mathingly Stanger Malur
Docket HIT-294

CERTIFIED COPY OF
PRIORITY DOCUMENT

２００１年 ７月２７日

特許庁長官
Commissioner,
Japan Patent Office 及川耕造

出証番号 出証特２００１－３０６６０３３

【書類名】　特許願

【整理番号】　NT00P1116

【提出日】　平成13年 3月 6日

【あて先】　特許庁長官　殿

【国際特許分類】　G06K 19/00

【発明者】

【住所又は居所】　東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内

【氏名】　神永　正博

【発明者】

【住所又は居所】　東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内

【氏名】　遠藤　　隆

【発明者】

【住所又は居所】　東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内

【氏名】　渡邊　高志

【発明者】

【住所又は居所】　東京都国分寺市東恋ヶ窪一丁目２８０番地　株式会社日立製作所　中央研究所内

【氏名】　大木　　優

【特許出願人】

【識別番号】　000005108

【氏名又は名称】　株式会社日立製作所

【代理人】

【識別番号】　100068504

【弁理士】

【氏名又は名称】　小川　勝男

【電話番号】　03-3661-0071

【選任した代理人】

【識別番号】　100086656

【弁理士】

【氏名又は名称】　田中　恭助

【電話番号】　03-3661-0071

【選任した代理人】

【識別番号】　100094352

【弁理士】

【氏名又は名称】　佐々木　孝

【電話番号】　03-3661-0071

【手数料の表示】

【予納台帳番号】　081423

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　耐タンパー暗号処理方法

【特許請求の範囲】

【請求項１】

（１）　ＲＳＡの公開鍵（ｅ，Ｎ；モジュラスＮは２つの素数ｐ，ｑの積）に対応する秘密鍵指数ｘを記憶装置に格納し、

（２）　入力手段を介して暗号メッセージyを入力し、

（３）　pまたはその倍数を法とするｙの剰余ｙｐ、およびqまたはその倍数を法とするｙの剰余ｙｑを計算し、

（４）　ｐ－１またはその倍数を法とするｘの剰余をｘｐ、ｑ－１またはその倍数を法とするｘの剰余をｘｑとするときに、ｐまたはその倍数を法とするｙｐ＾ｘｐの剰余であるＣｐを計算し、ｑまたはその倍数を法とするｙｑ＾ｘｑの剰余であるＣｑを計算するに際して、

（４ａ）下記（４ｂ）と（４ｃ）のいずれの処理を行うかを、ｘｐ、ｘｑを構成する少なくとも１ビットのビット列であるビットブロックの処理ごとに決定し、

（４ｂ）ｘｐの処理すべき前記ビットブロックについて所定のべき乗剰余計算を行なって、その計算結果を記憶装置に格納し、

（４ｃ）ｘｑの処理すべき前記ビットブロックについて所定のべき乗剰余計算を行なって、その計算結果を記憶装置に格納し、

（５）ｘｐ全体についてのＣｐの計算及びｘｑ全体についてのＣｑの計算を終了したとき、ＣｐとＣｑの差分に基づいてＲＳＡ復号化計算ｙ＾ｘｍｏｄ　Ｎを計算し、

（６）計算した前記ＲＳＡ復号化計算の結果を出力することを特徴とする耐タンパー暗号処理方法。

【請求項２】

前記ｙｐ，ｙｑ，ｘｐ，ｘｑについて、ｙｐ＝ｙｍｏｄ　ｐ，ｙｑ＝ｙｍｏｄ　ｑ，ｘｐ＝ｘｍｏｄ（ｐ－１），ｘｑ＝ｘｍｏｄ（ｑ－１）として計算することを特徴とする請求項１記載の耐タンパー暗号処理方法。

【請求項３】

生成した乱数を用いて前記（４b）と（４c）のいずれの処理を行うかを決定することを特徴とする請求項１記載の耐タンパー暗号処理方法。

【請求項４】

前記（４ａ）の処理を前記ｘｐ，ｘｑのビットパターンの一部に適用し、前記ビットパターンの残りの部分については、前記（４ｂ）と（４ｃ）のいずれか一方を行なった後に他方を実行することを特徴とする請求項１記載の耐タンパー暗号処理方法。

【請求項５】

（１）下記（２）と（３）のいずれのステップを選択するかを、一演算単位の処理をするごとに決定するステップと、

（２）メモリ中のデータＡのビットパターンの一演算単位を第１のレジスタＲ１に転送し、その後にメモリ中のデータＢのビットパターンの一演算単位を第２のレジスタＲ２に転送するステップと、

（３）前記データＢのビットパターンの一演算単位を前記第２のレジスタＲ２に転送し、その後に前記データＡのビットパターンの一演算単位を前記第１のレジスタＲ１に転送するステップと、

（４）前記第１のレジスタＲ１の内容と前記第２のレジスタＲ２の内容について所定の演算を行うステップと、

（５）前記演算の結果をメモリに格納するステップとを有し、

（６）前記データＡと前記データＢについての前記演算が終了するまで前記（１）から（５）までのステップを繰り返すことを特徴とする耐タンパー暗号処理方法。

【請求項６】

（１）下記（２）と（３）のいずれのステップを選択するかを、一演算単位の処理をするごとに決定するステップと、

（２）メモリ中のデータＡの一演算単位を第１のレジスタＲ１に転送し、その後にメモリ中のデータＢの一演算単位を第２のレジスタＲ２に転送するステップと、

（３）前記データＡの前記一演算単位を前記第２のレジスタＲ２に転送し、その

後に前記データＢの前記一演算単位を前記第１のレジスタＲ１に転送するステップと、

（４）前記第１のレジスタＲ１の内容と前記第２のレジスタＲ２の内容について所定の演算を行うステップと、

（５）前記演算の結果をメモリに格納するステップとを有し、

（６）前記データＡと前記データＢについての前記演算が終了するまで前記（１）から（５）までのステップを繰り返すことを特徴とする耐タンパー暗号処理方法。

【請求項７】

生成した乱数に対応して前記（２）と（３）のいずれの処理を行うかを決定することを特徴とする請求項６記載の耐タンパー暗号処理方法。

【請求項８】

前記所定の演算は、算術和「＋」をとる演算であることを特徴とする請求項６記載の耐タンパー暗号処理方法。

【請求項９】

前記所定の演算は、算術積「×」をとる演算であることを特徴とする請求項６記載の耐タンパー暗号処理方法。

【請求項１０】

前記所定の演算は、論理和ＯＲ、論理積ＡＮＤ、排他的論理和ＥＸＯＲのうちのいずれか１つであることを特徴とする請求項６記載の耐タンパー暗号処理方法。

【請求項１１】

（１）メモリ中のデータＡのビットパターンのうち、いずれかの未処理の一演算単位を選択するステップと、

（２）選択された前記データＡの前記一演算単位を第１のレジスタＲ１に転送するステップと、

（３）選択された前記データＡの前記一演算単位に対応するメモリ中のデータＢのビットパターンのうちの一演算単位を第２のレジスタＲ２に転送するステップと、

（４）前記第１のレジスタＲ１の内容と前記第２のレジスタＲ２の内容について所定の演算を行うステップと、

（５）前記演算の結果をメモリに格納するステップとを有し、

（６）前記データＡと前記データＢについての前記演算が終了するまで上記（１）から（５）までのステップを繰り返すことを特徴とする耐タンパー暗号処理方法。

【請求項１２】

生成した乱数に対応して前記未処理の前記一演算単位を選択することを特徴とする請求項１１記載の耐タンパー暗号処理方法。

【請求項１３】

前記所定の演算は、論理和ＯＲ、論理積ＡＮＤ、排他的論理和ＥＸＯＲのうちのいずれか１つであることを特徴とする請求項１１記載の耐タンパー暗号処理方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、高いセキュリティを持つＩＣカードなどに適用される耐タンパー暗号処理方法に関するものである。

【０００２】

【従来の技術】

ＩＣカードは、勝手に書き換えることが許されないような個人情報の保持や、秘密情報である暗号鍵を用いたデータの暗号化や暗号文の復号化を行う装置である。ＩＣカード自体は電源を持っておらず、ＩＣカード用のリーダライタに差し込まれると、電源の供給を受け、動作可能となる。動作可能になると、ＩＣカードは、リーダライタから送信されるコマンドを受信し、そのコマンドに従ってデータの転送等の処理を行う。ICカードの一般的な解説は、オーム社出版電子情報通信学会編 水沢順一著「ICカード」などにある。

【０００３】

ＩＣカードの構成は、図１に示すように、カード101の上にＩＣカード用チッ

プ102を搭載したものである。図に示すように、一般にＩＣカードは、ＩＳＯ７８１６の規格に定められた位置に供給電圧端子Vcc、グランド端子GND、リセット端子RST、入出力端子I/O、クロック端子CLKを持ち、これらの端子を通してリーダーライタから電源の供給やリーダライタとのデータの通信を行う(W.Rankl and Effing：SMARTCARD HANDBOOK、John Wiley & Sons、1997、pp.41参照)。

【０００４】

ＩＣカード用チップの構成は、基本的には通常のマイクロコンピュータと同じ構成である。その構成は、図2に示すように、中央処理装置（ＣＰＵ）201、記憶装置204、入出力（I/O）ポート207、コ・プロセッサ202からなる（コ・プロセッサはない場合もある）。ＣＰＵ201は、論理演算や算術演算などを行なう装置であり、記憶装置204は、プログラム205やデータを格納する装置である。入出力ポート207は、リーダライタと通信を行なう装置である。コ・プロセッサ202は、暗号処理そのもの、または暗号処理に必要な演算を高速に行なう装置であり、例えば、ＲＳＡ暗号の剰余演算を行うための特別な演算装置や、ＤＥＳ暗号のラウンド処理を行なう暗号装置などがある。ＩＣカード用プロセッサの中には、コ・プロセッサを持たないものも多くある。データバス203は、各装置を接続するバスである。

【０００５】

記憶装置２０４は、ＲＯＭ(Read Only Memory)やＲＡＭ(Random Access Memory)、ＥＥＰＲＯＭ(Electrical Erasable Programmable Read Only Memory)などからなる。ＲＯＭは、変更できないメモリであり、主にプログラムを格納するメモリである。ＲＡＭは自由に書き換えができるメモリであるが、電源の供給が中断されると、記憶している内容は消滅する。ＩＣカードがリーダライタから抜かれると電源の供給が中断されるため、ＲＡＭの内容は、保持されなくなる。ＥＥＰＲＯＭは、電源の供給が中断されてもその内容を保持することができるメモリである。書き換える必要があり、ＩＣカードがリーダライタから抜かれても、保持するデータを格納するために使われる。例えば、プリペイドカードでのプリペイドの度数などは、使用するたびに書き換えられ、かつリーダライタが抜かれて

もデータを保持する必要があるため、ＥＥＰＲＯＭで保持される。

【０００６】

ＩＣカードは、プログラムや重要な情報がＩＣカード用チップの中に密閉されているため、重要な情報を格納したり、カードの中で暗号処理を行うために用いられる。従来、ＩＣカードに適用される暗号を解読する難しさは、暗号アルゴリズムの解読の困難さと同じと考えられていた。しかし、ＩＣカードが暗号処理を行っている時の消費電流を観測し、解析することにより、暗号アルゴリズムの解読より容易に暗号処理の内容や暗号鍵が推定される可能性が示唆されている。消費電流は、リーダライタから供給されている電流を測定することにより観測することができ、この攻撃法及びその危険性の詳細は、John Wiley & sons社 W.Rankl & W.Effing著 「Smart Card Handbook」の8.5.1.1 Passive protective mechanisms(263ページ)に記載されている。

【０００７】

ＩＣカード用チップを構成しているＣＭＯＳは、出力状態が１から０あるいは０から１に変わった時に電流を消費する。特に、データバス203においては、バスドライバーの電流や、配線及び、配線に接続されているトランジスタの静電容量のため、バスの値が１から０あるいは０から１に変わると大きな電流が流れる。そのため、消費電流を観測すれば、ＩＣカード用チップの中で、何が動作しているか分かる可能性がある。

【０００８】

図3は、ＩＣカード用チップの１サイクルでの消費電流の波形を示したものである。処理しているデータに依存して、電流波形が301や302のように異なる。このような差は、バス203を流れるデータや中央演算装置201で処理しているデータに依存して生じる。

【０００９】

コ・プロセッサ202は、ＣＰＵと並列に、例えば512ビットの剰余演算を行うことができる。そのため、ＣＰＵの消費電流とは異なった消費電流波形の長時間の観測が可能である。その特徴的な波形を観測することにより、コ・プロセッサの動作回数を容易に測定することができる。コ・プロセッサの動作回数が暗号鍵と

何らかの関係があるならば、コ・プロセッサの動作回数から暗号鍵を推定できる可能性がある。

【００１０】

また、コ・プロセッサでの演算内容が、暗号鍵に依存した偏りがあると、その偏りが消費電流から求められ、暗号鍵が推定される可能性がある。

【００１１】

ＣＰＵでも同様の事情が存在する。暗号鍵のビット値は決まっているため、処理するデータを変更して、消費電流を観測することにより、暗号鍵のビット値の影響が観測できる可能性がある。これらの消費電流の波形を統計的に処理することにより、暗号鍵を推定できる可能性がある。

【００１２】

【発明が解決しようとしている課題】

本発明の課題は、ＩＣカード用チップでのデータ処理と消費電流との関連性を減らすことである。消費電流とチップの処理との関連性が減れば、観測した消費電流の波形からＩＣカードチップ内での処理や暗号鍵の推測が困難になる。本発明の着眼点の一つは、ＩＣカードチップで中国人剰余定理を用いたべき乗剰余演算の処理の順序をアタッカーに推定されないように変更することにより、消費電流の波形から、処理や暗号鍵の推測を困難にするものである。本発明のもう一つの着眼点は、算術和や算術積を二つのレジスタの値に対して実行する際に、レジスタへの転送順序をアタッカーに推定されないように変更することにより、消費電流の波形から、処理や暗号鍵の推測を困難にするものである。

【００１３】

本発明の目的は、上記の課題について、実現手段としての耐タンパー暗号処理方法を提供することにある。

【００１４】

【課題を解決するための手段】

ＩＣカードチップに代表される耐タンパー装置は、プログラムを格納するプログラム格納部、データを保存するデータ格納部を持つ記憶装置と、プログラムに従って所定の処理を実行しデータ処理を行う中央演算装置(ＣＰＵ)を持ち、プロ

グラムは、ＣＰＵに実行の指示を与える処理命令から構成される一つ以上のデータ処理手段からなる情報処理装置として捉えることができる。

【００１５】

本発明の第一は、中国人剰余定理（ＣＲＴ：Chinese Remainder Theorem）を用いてべき乗剰余計算の処理を行なう際に、処理しているデータとＩＣカード用チップの消費電流の関連性を減らす方法は、モジュラスを素因数に分離し、それぞれの素因数を法とするべき乗剰余計算の処理の順序を本来の処理順序とは異なる方式で処理する方法である。

【００１６】

中国人剰余定理とは、法Ｎによる剰余を、Ｎの互いに素な因数に対する剰余によって表現するものである。例えば、N = pqのようにモジュラスが二つの互いに素な素数の積である場合には、a mod Nを、a mod p と a mod qを用いて表現するのである。このように、モジュラスを分離して表現することにより、特にべき乗剰余計算の高速化が可能となる。すなわち、べき乗剰余計算y^x mod Nを、N=pqの場合に、y^x mod p、 y^x mod qを用いて表現すると、y^x mod p = (y mod p)^(x mod (p-1)) mod p、 y^x mod q = (y mod q)^(x mod (q-1)) mod qと書くことができるので、p、qが共に、N^(1/2)程度であれば、y、 xが、共に半分のデータ長となる。これにより、個々のべき乗剰余計算は、元の長さの処理時間の約1/8の時間で計算することができる。この高速化効果は非常に大きいものである。

【００１７】

一方、べき乗剰余計算は、指数のビットパタンを順に読んで実行される。よく知られたアルゴリズムであるアディション･チェイン（加法連鎖）方式やスライディング･ウインドウ方式では、指数ビットを上位（又は下位）から順にビット値を取り出し、二乗剰余演算と剰余乗算演算を組み合せて処理を行なう。この二つのべき乗剰余計算の途中の処理を本来の処理順序とは異なる方式で処理する。例えば、通常は、yp^xp mod pの指数のビットに応じる単位処理をビット長さ分だけ続け、その後、yq^xq mod qの処理を行なう。

【００１８】

本発明に基づくべき乗剰余処理においては、yp^xp mod pの途中処理 → yq^xq mod pの途中処理 →yq^xq mod qの途中処理 → yp^xp mod pの途中処理 → yp^xp mod pの途中処理 → yp^xp mod pの途中処理 → yq^xq mod qの途中処理 →...→ などのように、両者の処理とは異なる処理順序で実行する。 これにより、電流観測を行なっているアタッカーは混乱する。処理順序を定められた単純な規則を用いずに、ランダム（疑似ランダム）に行なうことにより、効果をさらに高めることができる。特に、電流波形の平均化を行なうことによってノイズを除去する通常の波形観測を著しく困難にすることができる。

【００１９】

また、本発明をxp = (x mod (p-1)) + k(p-1)、 xq = (x mod (q-1)) + j(q-1)（k、 jは乱数）と表現する方式や、モジュラスをp、qそのものではなく、その乱数倍にするなどの操作を組み合せることで、さらに効果を高めることができる。

【００２０】

本発明の第二は、マイクロコンピュータに実装された二項演算、例えば、算術和「＋」、算術積「×」や、論理和、論理積、排他的論理和を実行する際に必要な転送処理において、その転送処理の順序を毎回変更するものである。例えば、ＲＡＭに格納されたデータAとBに対して、ソースレジスタRs、 ディスティネーションレジスタRdの値の算術和を行なう際、通常は、Aの値をRs（又はRd）、Bの値をRd（又はRs）に転送した後、ADD Rs、 Rd（これは、アセンブラ形式で、RsとRdの算術和を計算し、Rdに結果を格納することを意味する）を実行するが、その際、転送の順序は固定されている。本発明では、Aの値をRsに転送した後、Bの値をRdに転送するか、Bの値をRdに転送した後、Aの値をRsに転送するかをアタッカーが予測できないように入れ換えることにより、電流観測を行なっているアタッカーは混乱する。処理順序を定められた単純な規則を用いずに、ランダム（疑似ランダム）に行なうことにより、効果をさらに高めることができる。特に、電流波形の平均化を行なうことによってノイズを除去する通常の波形観測を著しく困難にすることができる。

【００２１】

【発明の実施形態】

実施例を説明する準備として、ＣＲＴを用いたＲＳＡ暗号のべき乗剰余計算の基本アルゴリズム及び、べき乗剰余計算の代表的な方式であるアディション･チェイン方式及びスライディング･ウインドウ方式について説明する。

【００２２】

ＲＳＡ暗号では、大きな素数、例えば512ビットの２つの素数p、qの積 N = pq と Nと互いに素な数 e（ICカードでは、3や、65537が用いられることが多い）をとり、これを公開鍵として公開鍵簿に登録する。このとき、この公開鍵の持ち主Ａに送信者Ｂは、1以上Ｎ－１以下の数で表現されたデータ（平文）Ｍ を、

y = M^e mod N

として暗号化して送信する。ここで、M^eはMのe乗を表す記号とする。

この暗号文Ｒ を受け取った持ち主Ａは、xe mod (p-1)(q-1) = 1となる秘密鍵 x を用いて

y^x mod N

を計算する。ここで、(p-1)(q-1)は、Nのオイラー関数の値f(N)である。これは、Nと互いに素な自然数の個数に等しい。オイラーの定理によれば、

y^((p-1)(q-1)) mod N = 1

が成り立つ。一方、xe = 1 + k(p-1)(q-1)（kは整数）と書くことができるので、

y^x mod N
= (M^e)^x mod N
= M^(ex) mod N
= M^(1+k(p-1)(q-1)) mod N
= M*M^(k(p-1)(q-1)) mod N
= M

が成り立つ。従って、y^x mod Nを計算することで、持ち主Ａは、送信者Ｂの平文 Ｍ を復号することができる。この際、秘密鍵 x を計算するのに、Nの素因数 p、 q が用いられている。現在のところ、素因数分解を介さないで、xを計算する方法は知られておらず、大きな素数の積を因数分解することは、現実的でない

時間が必要であるので、Nを公開しても持ち主Ａの秘密鍵は安全である。

【００２３】

ＩＣカードでは、公開鍵指数eとして、3や65537が用いられることが多い。これは暗号化の計算時間を短縮するという意味もあるが、eの値を知っても直接的に秘密鍵指数ｘやNの素因数が危険に曝されることはないという事情によるものである。

【００２４】

この計算法としては、アディション･チェイン方式などが採用される（上記「暗号理論入門」参照）ことが多いが、このようなアルゴリズムでは処理が遅く、ＩＣカードを用いたトランザクションに要する時間がユーザの許容範囲を超えてしまう可能性がある。そこで、単純にｘ、Nに対するべき乗剰余計算を行なわずに、公開モジュラスNの二つの素因数p、qに対するべき乗剰余計算結果からＭを導く方法がＣＲＴである。

【００２５】

図４を用いてＣＲＴの処理を簡単に説明する。まずカード101は、計算に用いる値k=p^(-1) mod q、 xp = x mod (p-1)、 xq = x mod (q-1)の値を計算する（ステップ401）。ここでp-1，q-1の代わりに各々の倍数でもよい。普通、これらの値はＥＥＰＲＯＭに格納しておく。次にＩ／Ｏポートから暗号文yを受け取り（ステップ402）、この暗号文yから素因数p，qまたはp，qの倍数を法とする剰余yp = y mod p、 yq = y mod qを求め、これをＲＡＭに格納する（ステップ403）。次に、二つのべき乗剰余計算：

$$Cp = yp^{xp} \bmod p,\quad Cq = yq^{xq} \bmod q$$

を行なう（ステップ404， 405）。次に、再結合計算：

$$S = (Cq - Cp)*k \bmod q$$

$$M = S*p + Cp$$

を行ない（ステップ406，407）、Ｉ／Ｏポートに平文Ｍを出力する（ステップ408）。このＭが実際の$y^x \bmod N$に一致する。

【００２６】

この事実を数値的に確認しておく。暗号文y = 79、 N = 187(=11*17)、 x = 1

07とする。このｘは、Nのオイラー関数値(11-1)*(17-1)=160に関して、ｅ = 3の逆数になっている。このとき、実際の値は、

M = 79^107 mod 187

= 79^(5*3*7 + 2) mod 187

= 79^2 * (79^5 mod 187)^(3*7) mod 187

= 79^2 * 10^(3*7) mod 187

= 79^2 * (10^3 mod 187)^7 mod 187

= 79^2 * (65^7 mod 187) mod 187

= 79^2 * 142 mod 187

= 29

である。

【００２７】

これをＣＲＴを用いて計算する。11*14 mod 17 = 1であるから、k = 11^(-1) mod 17 = 14であり、xp = 107 mod (11-1) = 7、 xq = 107 mod (17-1) = 11である。また、yp = 79 mod 11 = 2、 yq = 79 mod 17 = 11となる。

【００２８】

Cp = 2^7 mod 11 = 7

Cq = 11^11 mod 17 = 12

となるので、

S = (12 - 7)*14 mod 17 = 2

M = 2*11 + 7 = 29

となり、先の値と一致する。

【００２９】

ＣＲＴを用いると高速化できる理由は、べき乗剰余計算では、データ長さの3乗に比例して、計算量が増加するのに対して、ＣＲＴでは、データの長さが半分のものを二つ計算するため、それぞれのべき乗剰余計算の計算量は、1/8で済み、これを二回実行しても、両者の計算量の合計は、1/8 * 2 = 1/4で済むからである。実際には、データの変換や、再結合計算を行なう必要があるので4倍速にはならないが、速度は3倍程度に向上する。

【００３０】

次に、図５を参照してべき乗剰余計算のアディション･チェイン方式による計算アルゴリズムを説明する。この方式は、最も多く利用されているものである。ここでは、y^x mod Nの計算を秘密鍵 x のビットを２ビット毎に区切り、上位から読んでいき、それが、00、01、10、11のいずれであるかに応じて、y[0] = 1、y[1] = y、 y[2] = y^2 mod N、 y[3] = y^3 mod N を対応させ（ステップ501）、剰余乗算計算を行うことによって実現したものである。 勿論、２ビット毎に区切ることは説明の便宜のためであり、実際には１ビット、３ビット、４ビットをまとめて計算することもある。その際も考え方は全く同じである。

この計算の際、４乗の計算（ステップ502）は、ｘのビットと無関係に行われるが、その次にカード101が行う剰余乗算計算においては、xのビット（２ビット分を指す）値に応じて、条件分岐（ステップ503，504，505，506）を行い、それぞれ、ステップ507，508，509，510の剰余乗算計算を行う。

この方式で正しい計算が行なえることを簡単に数値例で確認する。この計算方式の本質的部分は指数部分であるので、例として、指数部分のみ数値である

S = y^219 mod N

の例を取り挙げる。ここで、219は二進数表示で11011011である。これをもとに、２ビット幅のアディション･チェイン方式で計算する。ステップ501の処理に相当する11011011を2ビットブロックに分割すると、11 01 10 11である。S = 1（初期化）を行ない、これをＮを法として4乗する。勿論、1を4乗しても1である。次に指数部の先頭のビットブロックを読む。これは11であるから、y[3] = y^3 mod Nを乗じて、S = y^3 mod Nとなる。次に、再びループし、これをＮを法として4乗すると、y^12 mod Nとなる。指数部のビットブロックの先頭から2番目のビットブロックを読むと、これは、01であるから、y[1] = y を乗じて、S = y^13 mod Nとなる。再びループし、これをＮを法として4乗すると、S = y^52 mod Nとなる。指数部のビットブロックの先頭から3番目のビットブロックを読むと、これは、10であるから、y[2] = y^2 mod N を乗じて、S = y^54 mod Nとなる。再びループし、これをＮを法として4乗すると、S = y^216 mod Nとなる。指数部のビットブロックの先頭から4番目のビットブロックを読むと、これは、11である

から、y[1] = y^3 mod N を乗じて、S = y^219 mod Nとなる。これが求める答である。

【００３１】

次に、もう一つの代表的なべき乗剰余計算アルゴリズムであるスライディング・ウインドウ方式を説明する。スライディング・ウインドウ方式では、例えば、次のように処理を行なう。

【００３２】

S = y^2226 mod N

を計算するにあたり、テーブルy[2] = y^2 mod Nおよび y[3] = y^3 mod Nを用意する。この際、y[0]、 y[1]は不要である。2226は、２進法で、100010110010と書くことができる。これを上位より読んで、2ビットが1を上位に持っているときはそれを一塊とみなし、0が続いているときは2乗剰余演算を行なうとみなす。つまり、100010110010 = 10 0 0 10 11 0 0 10 という分解に対応するように処理する。従って、S=1と初期化した後、まず、10に対応する処理、すなわち、Sの4乗剰余演算を行なって、Sにy[2]を掛ける。このとき、S = y[2] = y^2 mod Nとする。次に、0に対応した処理、すなわち2乗剰余演算を行なって、S = y^4 mod Nを得る。続く0に対応する処理、S = y^8 mod Nを行ない、次に、10に対応する処理を行なえば、S = ((y^8 mod N)^4 mod N * y^2 mod N) mod N = y^34 mod Nとなる。続く処理は、11に対応した、S = ((y^34 mod N)^4 * y^3 mod N) mod N = y^139 mod Nを実行する。さらに続く２つの0に対応して4乗剰余演算を行ない、S = (y^139 mod N)^4 mod N = y^556 mod Nとし、最後の10に対応する処理S = ((y^556 mod N)^4 mod N * y^2 mod N) mod N = y^2226 mod Nによって、求める答を得る。この方式は、テーブルとして、指数ビットの先頭が1であるものだけを利用するので、ＲＡＭの容量が半分で済むという利点がある。

【００３３】

上記のアディション・チェイン方式、スライディング・ウインドウ方式は、モンゴメリ法と呼ばれる方式を用いて実行されることがあることに注意しておく。モンゴメリ法とは、剰余乗算演算 AB mod Nを高速に実行する方式であり、特にハードウエア化に向いた方法である。簡単にそのしくみを説明する。詳細は、モン

ゴメリの原著論文”Modular Multiplication Without Trial Division”、 Mathematics of Computation 44、 170、 pp.519-521(1985))に記載されている。

【００３４】

モンゴメリ法の本質は、殆ど全てのコンピュータにおいて、mod 2^nの演算は、上位ビットを無視することにより実現できるということを利用することにある。すなわち、AB mod Nの計算を2のべきを法とする演算に置き換えることにその本質がある。ＲＳＡ暗号では、Nは、大きな素数の積であるから奇数であり、従って任意の２のべきと互いに素である。そこで、M、 Wを未知数とした不定方程式：

$$AB + MN = WR$$

を考える。ここで、A、 Bのビット長はnであるものとし、R = 2^nとする。このとき、この不定方程式は無限個の解を有する。このようなMを見つければ、WはABR^(-1) mod Nと合同である。MはRおきに並んでいるので、Rより小さい非負の値がとれる。このとき、WはABR^(-1) mod Nであるかまたは、ABR^(-1) mod N+Nである。後者の場合は、Nを引き算して求める答を得る。

【００３５】

このように、モンゴメリ法においては、ABR^(-1) mod Nの形で演算が行なわれるので、上記のアルゴリズム、例えばアディション･チェイン方式においては、図５のステップ501で実行されるテーブル作成処理では、それぞれのテーブルの値をy[0] = R mod N、 y[1] = yR mod N、 y[2] = y^2R mod N、 y[3] = y^3R mod Nに置き換えて処理する。Sの初期値もR mod Nとする。すると、被乗数Aと乗数Bの値は、もとの値のR倍になっているため、ABR^(-1) mod Nの処理では、必ずR倍の項が残ることになる。このデータ形式をモンゴメリフォーマットと呼ぶことにすれば、アディション･チェイン、スライディング･ウインドウ両方式を、モンゴメリフォーマットで実行し、最後に、R^(-1) mod N倍することにより、求める答を得ることができる。モンゴメリ法を用いた処理においても、本発明の方式は、本質的に何ら変更を必要としない。

上記の準備の下で、本発明の実施例を説明する。図６及び図７を用いて、べき乗剰余計算の方式として、モンゴメリ法を用いた剰余乗算演算を用い、2ビットア

ディション･チェイン方式を用いる場合について説明する。ＲＳＡ暗号系において、モジュラスN、秘密鍵指数xによって、暗号文ｙを復号する処理M = y^x mod N (N = pq)を行なうに際し、まずＣＲＴに用いる前処理計算（ステップ601）において、k = p^(-1) mod q、 xp = x mod (p-1)、 xq = x mod (q-1)を準備し、ＥＥＰＲＯＭに格納しておく。暗号文yを取得し（ステップ602）、2ビットのモンゴメリフォーマットに変換されたテーブルを用意する（ステップ603）。このテーブルは、Nの素因数p、qをモジュラスとしたもの両方からなる。すなわち、R mod p、 yR mod p、 y^2R mod p、 y^3R mod p、 R mod q、 yR mod q、 y^2R mod q、 y^3R mod qを用意する。このテーブルは、ＣＲＴにおける二つのべき乗剰余処理、Cp = (y mod p)^xp mod p、 Cq = (y mod q)^xq mod qの2ビットアディション･チェイン処理に必要なものである。この二つのべき乗剰余処理が終わったかどうかをステップ604、605の条件分岐処理で判定する。ステップ604の条件分岐処理は、xp、xq全てのビット処理が終了したかどうかを判定するものであり、全てのビットが終了していれば、ステップ610の処理に移る。ステップ610、611では、Cp、 Cqをモンゴメリフォーマットから通常のフォーマットに書き直す処理Cp = Cp*R^(-1) mod p、Cq = Cq*R^(-1) mod qを行ない、ステップ612では、S = (Cq - Cp)*k mod qが求められ、続くステップ613の処理で、M =S*p + Cpを求め、最後にこの値Mを出力する。一方、xp、 xqのうち、少なくとも一方が終了していない場合、ステップ604の条件分岐はスルーし、ステップ605の条件分岐処理に入る。ステップ605では、xp、 xqのどちらが終了しているか、または終了していないかが判定される。いずれも終了していない場合、ステップ606の処理に進む。ステップ606では、1ビットの乱数vが生成され、v = 0であれば、Cpの処理（ステップ608）、v = 1であれば、Cqの処理（ステップ609）を行ない、終了後、再びステップ604の処理に移る。

ここでCpの処理、Cqの処理とは、図８に示した処理である。ここでは、Cpの場合が書かれているが、Cqの場合でも処理内容は同一である。図８の処理は、まず、４乗剰余乗算処理を行なう。モンゴメリ法で実行するので、R^(-1) mod pというファクターが含まれているが、本質的には、通常の４乗剰余乗算処理と考えてよい。次に指数部の2ビットブロックを受け取り（ステップ702）、このビットブロ

ックが、00であるか、01であるか10であるか、11であるかに応じて、それぞれステップ707、708、709、710の処理に分岐する（ステップ703、 704、 705、 706）。ステップ707、708、709、710の処理は、乗数が異なるのみで、他は同一の処理であるから、ステップ705からの分岐の場合のみ説明する。このときは、y^2R mod pを乗じて、モンゴメリ剰余乗算処理を行ない、処理が終了すれば、Cpの値をＲＡＭに書き戻す（ステップ711）。他も同様である。ステップ605において、xpが終了していれば、Cpの処理は終了しているので、Cqの処理（ステップ609）を実行する。逆に、xqが終了していれば、Cqの処理が終了していることになるので、Cpの処理（ステップ608）を実行する。この処理で、正しいCp，Cqの値が得られることは明白である。

本実施例に基づいて動作するプログラムの実行時の消費電力を解析するアタッカーは、通常の処理系列とは異なる電流パターンを見ることになる。特に本実施例では、処理系列をランダムに選択するため、実行のたびに異なる処理系列を観測する。例えば、本来のxpのビットパターンが、10 11 00 10であり、xqのビットパターンが、11 00 01 11であったと仮定する。このとき、通常は、Cpの処理を行なってから、Cqの処理を行なう（又はCqの処理を行なってから、Cpの処理を行なう）。従って、通常のＣＲＴによる処理で観測される処理系列は、10 11 00 10 11 00 01 11となる。一方、本実施例のＣＲＴ処理系列は、毎回異なる処理系列を生成する。例えば、分岐用の乱数がv = 0 1 1 0 0 1 0 1 であれば、10 11 00 11 00 01 10 11となり、v = 1 1 0 0 0 1 1 0であれば、11 00 10 11 00 01 11 10となる。通常処理は、v = 0 0 0 0 1 1 1 1に対応する。並べて比較してみると、

通常CRTの処理	10 11 00 10 11 00 01 11
本実施例の処理１	10 11 00 11 00 01 10 11
本実施例の処理２	11 00 10 11 00 01 11 10

のようになる。このように、ランダムに攪拌された処理系列から、元の処理系列を推測することは困難となる。また実際の電流波形観測においては、複数の波形を各時点毎に平均化して、ノイズを除去することが多い。（平均化でノイズを除去する方法は、確率論における大数の法則に基づいている。）この場合、処理順

序の撹乱の効果は、波形が異なるビット値に対する波形の平均として観測され、指数による特徴が消失することによりさらに高まるものと考えられる。

【００３６】

上記実施例においては、ビットパタンの一処理単位とは、xp、 xqの2ビットブロック00、 01、 10、 11に相当する。これを、1ビット、3ビット、4ビット等々のブロックとしても、上記処理がビット単位を変えるのみで、同一の構造を持つことは明らかであろう。

【００３７】

また本実施例では、乱数ｖを用いて条件分岐処理をスイッチしているが、これを線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係である。

【００３８】

次に本発明の「ビットパタンの一処理単位」のもう一つの側面を示すために、図９及び図１０を用いてスライディング･ウインドウ方式の場合を説明する。本質的には、図6及び図７の処理に類似しているが、指数ビットの扱いが若干異なる。比較を容易にするために、0以外の処理ビットの単位を2ビットとしたものを示す。

ＲＳＡ暗号系において、モジュラスN、秘密鍵指数xによって、暗号文ｙを復号する処理M = y^x mod N (N = pq)を行なうに際し、まず、ＣＲＴに用いる前処理計算（ステップ801）において、k = p^(-1) mod q、 xp = x mod (p-1)、 xq = x mod (q-1)を準備し、ＥＥＰＲＯＭに格納しておく。暗号文yを取得し（ステップ802）、アディション･チェイン方式の場合の00、01に相当するテーブル値を除いたモンゴメリフォーマットテーブルを用意する（ステップ803）。このテーブルは、Nの素因数p、qをモジュラスとしたもの両方からなる。すなわち、y^2R mod p、 y^3R mod p、 y^2R mod q、 y^3R mod qを用意する。このテーブルは、ＣＲＴにおける二つのべき乗剰余処理、Cp = (y mod p)^xp mod p、 Cq = (y mod q)^xq mod qの2ビットスライディング･ウインドウ処理に必要なものである。この二つのべき乗剰余処理が終わったかどうかをステップ804、805の条件分岐処理で判定する。ステップ804の条件分岐処理は、xp、xq全てのビット処理が終了した

かどうかを判定するものであり、全てのビットが終了していれば、ステップ810の処理に移る。ステップ810、 811の処理では、Cp、 Cqをモンゴメリフォーマットから通常のフォーマットに書き直す処理Cp = Cp*R^(-1) mod p、Cq = Cq*R^(-1) mod qを行ない、ステップ812では、S = (Cq - Cp)*k mod qが求められ、続くステップ813の処理で、M = S*p + Cpを求め、最後にこの値Mを出力する。一方、xp、 xqのうち、少なくとも一方が終了していない場合、ステップ804の条件分岐はスルーし、ステップ805の条件分岐処理に入る。ステップ805では、xp、 xqのどちらが終了しているか、または終了していないかが判定される。いずれも終了していない場合、ステップ806の処理に進む。ステップ806では、1ビットの乱数vが生成され、v = 0であれば、Cpの処理（ステップ808）、v = 1であれば、Cqの処理（ステップ809）を行ない、終了後、再びステップ804の処理に移る。ここで、Cpの処理（ステップ808）、Cqの処理（ステップ809）とは、図１１に示した処理である。ここでは、Cpの場合が書かれているが、Cqの場合でも処理内容は同一である。

図１１の処理は、まず、２乗剰余乗算処理を行なう。モンゴメリ法で実行するので、R^(-1) mod pというファクターが含まれているが、本質的には、通常の２乗剰余乗算処理と考えてよい（アディション･チェイン方式では、無条件に４乗剰余乗算処理を行なったがスライディングウインドウ方式ではこの点が異なっていることに注意）。次に指数部の1ビットを受け取り（ステップ902）、その値を読み、これが1であるかどうか判定し（ステップ903）、１でなければ、直接ステップ909の処理に移る。もしステップ903において、ビット値が1であれば、ステップ904にて、モンゴメリ法での２乗剰余乗算処理を行なう。次に、xpの次の1ビットが読めるかどうか（ビットの終端に来ているかどうか）を判定し（ステップ905）、そのビット値が0であるかどうか判定し（ステップ906）、0であれば、モンゴメリフォーマットでの乗算Cp=Cp*(y^2R mod p)*R^(-1) mod pを実行し（ステップ907）、0でなければ1であるので、Cp=Cp*(y^3R mod p)*R^(-1) mod pを実行し（ステップ908）、CpをＲＡＭに格納して（ステップ909）、剰余乗算処理を終了する。Cqの場合の処理も全く同様である。ステップ805において、xpが終了していれば、Cpの処理は終了しているので、Cqの処理（ステップ809）を実行する

。逆に、xqが終了していれば、Cqの処理が終了していることになるので、Cpの処理（ステップ808）を実行する。この処理で、正しいCp，Cqの値が得られることは明白である。

本実施例に基づいて動作するプログラムの実行時の消費電力を解析するアタッカーは、通常の処理系列とは異なる電流パターンを見ることになる。特に本実施例では、処理系列をランダムに選択するため、実行のたびに異なる処理系列を観測することになる。この事情を前実施例と同様の例で説明する。すなわち、xpのビットパターンが10 11 00 10であり、xqのビットパターンが11 00 01 11であったと仮定する。このとき、通常はCpの処理を行なってから、Cqの処理を行なう（又はCqの処理を行なってから、Cpの処理を行なう）。従ってスライディングウインドウ方式を用いた通常のＣＲＴによる処理で観測される処理系列は、10 11 0 0 10 11 0 0 0 11 1となる。一方、本実施例のＣＲＴ処理系列は、毎回異なる処理系列を生成する。例えば分岐用の乱数がv = 0 1 1 0 0 1 0 1 0 1であれば、10 11 0 11 0 0 0 0 10 11 1となり、v = 1 1 0 0 0 1 1 0 1 0 1であれば、11 0 1 0 11 0 0 0 0 11 10 1となる。通常処理は、v = 0 0 0 0 1 1 1 1に対応する。並べて比較してみると、スライディングウインドウ方式では、

通常CRTの処理　10 11 0 0 10 11 0 0 0 11 1

本実施例の処理１　10 11 0 11 0 0 0 0 10 11 1

本実施例の処理２　11 0 10 11 0 0 0 0 11 10 1

のようになる。このように、ランダムに攪拌された処理系列から元の処理系列を推測することは困難となる。

【００３９】

上記実施例においては、ビットパタンの一処理単位とは、xp、 xqのアディション･チェインのときとは異なっている。すなわちスライディングウインドウ方式の場合、ビットパタンの一処理単位は、「0」、「10」、「11」と考えることができる。さらにビット幅を増やして、「0」、「100」、「101」、「110」、「111」のように取ることもできる。同様にビットの幅を増やすことも容易である。このように、本発明でいう「ビットパタンの一処理単位」とは、同じ長さを持つビットブロックのみを意味するのではなく、スライディングウインドウ方式のと

きのように、異なる長さのものをも許容するものである。

【００４０】

また本実施例では、乱数ｖを用いて条件分岐処理をスイッチしているが、これを線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係である。

【００４１】

上記実施例は、Cp、 Cqの処理を同一の方式で行なっているが、これを変えることも本発明の趣旨を損ねない。例えば、Cpの処理系列を2ビット幅のアディション･チェイン方式で実行し、Cqの処理系列を上記の2ビットスライディング･ウインドウ方式で行なってもよい。この他、Cpの処理系列を2ビット幅のアディション･チェイン処理で実行し、Cqの処理系列を3ビット幅のアディション･チェイン方式で行なってもよい。以下、これら全ての場合について例を挙げることはしないが、このような場合に対して、本発明を適用することは、自然なことである。

【００４２】

次に、上記二つの実施例は、xp、 xq全てのビットに関して、順序撹乱を施しているが、これを一部のビット列に関してのみ適用することができることを示す。

【００４３】

これを説明するために、まず、「xの一部のビット列」を明確化する。そのために、いくつか例を挙げる。図１２は、xの二進数表示(x[n-1]x[n-2]...x[1]x[0])を分割したもので、1002は、「xの一部のビット列」の一例である。図１３は、xを二つの部分に区切ったものであり、1101は、「xの一部のビット列」の一例である。図１４は、図１３と同様にxを分割したものであり。1202は、「xの一部のビット列」の一例になっている。図１５は、xを5つの部分に分割したものである。1302と1304を合わせた部分は、「xの一部のビット列」の一例である。

【００４４】

上記の例に見るように、「xの一部のビット列」とは、「xのビットの連続した一部分の和集合」とみなすことができる。容易に類推できるように、「xの一部

のビット列」の取り方は上記の例の他に数多くある。以下、簡単のためxpの一部のビット列Bpなどという書き方をする。

【００４５】

ＣＲＴでの処理における指数xpの一部のビット列Bp、 指数xqの一部のビット列Bqを図１６のように定める。但し、ここでxp = (xp[n-1] xp[n-2] ...xp[1] xp[0])の各成分は2ビットブロックであるものとする。すなわち、xp[k]は、二進数00、 01、 10、 11のうちのいずれかである。従って、nはブロック数であり、例えば、xpが512ビットであれば、ブロック数は、512/2 = 256である。xqについても同様である。

【００４６】

ＲＳＡ暗号系において、モジュラスN、秘密鍵指数xによって、暗号文ｙを復号する処理M = y^x mod N (N = pq)を行なうに際し、まず、図１７～図１９に示すＣＲＴに用いる前処理計算（ステップ1501）において、k = p^(-1) mod q、 xp = x mod (p-1)、 xq = x mod (q-1)を準備し、ＥＥＰＲＯＭに格納しておく。暗号文yを取得し（ステップ1502）、2ビットのモンゴメリフォーマットに変換されたテーブルを用意する（ステップ1503）。このテーブルは、Nの素因数p、qをモジュラスとしたもの両方からなる。すなわち、R mod p、 yR mod p、 y^2R mod p、 y^3R mod p、 R mod q、 yR mod q、 y^2R mod q、 y^3R mod qを用意する。このテーブルは、ＣＲＴにおける二つのべき乗剰余処理、Cp = (y mod p)^xp mod p、 Cq = (y mod q)^xq mod qの2ビットアディション･チェイン処理に必要なものである。

【００４７】

次に、カウンタ conutをn-1にセットする（ステップ1504）。次にxp[count]に対応したモンゴメリフォーマットによる剰余乗算処理を行う（ステップ1505）。この処理は、図８に示すものと本質的に同一の計算である。つまり、Cpの場合、4乗剰余乗算を行なった後、xp[count]が、00、 01、 10、 11のいずれかに応じ、それぞれ、R mod p、 yR mod p、 y^2R mod p、 y^3R mod pをモンゴメリ法を用いた剰余乗算処理を行なうものである。ステップ1505の処理の後、CpをＲＡＭに格納し（ステップ1506）、カウンタcountを1減らす（ステップ1507） 。ステ

ップ1508の条件分岐処理においては、カウンタcountが、j-2になっていなければ、再び、ステツプ1505の処理に移り、countがj-2になれば、Cqの処理に移り、カウンタcountを再びn-1にセットする（ステップ1509）。次にxq[count]に対応したモンゴメリフォーマットによる剰余乗算処理を行なう（ステップ1510）。この処理は、図８に示すものと本質的に同一の計算である。ステップ1510の処理終了後、CqをＲＡＭに格納する。いちいち確認しないが、CpとCqの格納位置は重複してはならない。これは、CpとCqが互いに上書きしてしまうことを防ぐためである。以下、異なる変数名（定数名）で呼ばれているものについては、互いにＲＡＭ上で重複がないものと仮定して話を進める。

【００４８】

次に、本発明の処理をBp、 Bqに適用している部分の説明を行なう。この部分は、既にこれまで説明してきたものであるが、図に合わせて説明を再記する。まず、Bp、 Bqのビットブロックの処理が終了したかどうかを判定する（ステップ1514）。共に終了していれば、ステップ1520の処理に移る。もし、両方が終了していないならば、Bpか、Bqの少なくともいずれか一方は終了していない。ステツプ1515の条件分岐処理では、Bp、 Bqのどちらが終了しているか、あるいは共に終了していないかを判定する。いずれも終了していない場合、ステップ1516の処理に進む。ステップ1516では、1ビットの乱数vが生成され、ステップ1517の条件分岐処理にて、v = 0であれば、Cpの処理（ステップ1518）、v = 1であれば、Cqの処理（ステップ1519）を行い、終了後、再びステップ1514の処理に移る。ここで、Cpの処理（ステップ1518）、Cqの処理（ステップ1519）とは、図８に示した処理である。ここでは、Cpの場合が書かれているが、Cqの場合でも処理内容は同一である。ステップ1515において、Bpが終了していれば、Cpの処理は終了しているので、Cqの処理（ステップ1519）を実行する。逆に、Bqが終了していれば、Cqの処理が終了していることになるので、Cpの処理（ステップ1518）を実行する。

【００４９】

順序撹乱処理後、ステップ1520に移る。ここでは、カウンタcountをk-2にセットする。ここでkはx[k]のkであり、ステップ1532のkとは別の変数とする。次にxp[count]に対応したモンゴメリフォーマットによる剰余乗算処理を行なう（ステ

ップ1521）。この処理は、図８に示すものと本質的に同一の計算である。つまり、Cpの場合、4乗剰余乗算を行なった後、xp[count]が、00、 01、 10、 11のいずれかに応じ、それぞれR mod p、 yR mod p、 y^2R mod p、 y^3R mod pをモンゴメリ法を用いた剰余乗算処理を行なうものである。ステップ1521の処理の後、CpをＲＡＭに格納し（ステップ1522）、カウンタcountを1減らす（ステップ1523）。ステップ1524の条件分岐処理においては、カウンタcountが負になっていなければ、再びステップ1521の処理に移り、countが負になれば、Cqの処理に移り、カウンタcountを再びk-2にセットする（ステップ1525）。次にxq[count]に対応したモンゴメリフォーマットによる剰余乗算処理を行う（ステップ1526）。この処理は、図8に示すものと本質的に同一の計算である。ステップ1526の処理終了後、CqをＲＡＭに格納する（ステップ1527）。次にカウンタcountを1減らす（ステップ1528）。ステップ1529の条件分岐処理においては、カウンタcountが負になっていなければ、再びステップ1526の処理に移り、countが負になれば、ステップ1530の処理に移る。ステップ1530、1531では、Cp、 Cqをモンゴメリフォーマットから、通常のフォーマットに書き直す処理Cp =Cp*R^(-1) mod p、Cq = Cq*R^(-1) mod qを行ない、ステップ1532では、S = (Cq Cp)*k mod qが求められ、続くステップ1533の処理で、M = S*p + Cpを求め、最後にこの値Mを出力する。この処理で正しいMの値が得られることは明白である。本実施例に基づいて動作するプログラムの実行時の消費電力を解析するアタッカーは、通常の処理系列とは異なる電流パターンを見ることになる。特に本実施例では、処理系列をランダムに選択するため、実行のたびに異なる処理系列を観測する。

【００５０】

上記実施例においては、Bp、 Bqの位置と大きさが同一であったが、これは本発明の本質とは無関係である。すなわちBp、 Bqの位置や大きさを変えることができ、さらに、CpとCqの処理を一方はアディション･チェイン、他方はスライディング･ウインドウ方式といった変則的な適用も可能である。

【００５１】

また上記実施例では、図１２に示す比較的単純な位置をプロテクトしたが、先に「指数の一部のビット列」の定義部で述べたように、図１５に示すようなより

複雑な位置をプロテクトすることも可能である。

【００５２】

次に、他の実施例について説明する。実施例の説明に先立ち、背景を簡単に説明する。

【００５３】

一般に、マイクロコンピュータにおいては、中央演算装置（またはコ･プロセッサ等の演算装置）がレジスタと呼ばれる一時記憶領域を持っている。図２０は、マイクロコンピュータの中央演算装置と、ＲＡＭ（データ記憶装置）の図である。ＲＡＭ（1601）は、書換え可能なメモリである。このメモリ内に演算に用いる各々1602、1603で示すデータA、Bが格納されているのが一般的である。定数として扱う場合には、ＲＯＭやＥＥＰＲＯＭ等に格納されることもある。1604で示すデータCは演算結果を格納する領域であり、ＲＡＭ1601自体が、データバス1606、1607及びライトバス（書き込み用バス）1608に接続されており、ＲＡＭのデータを読み込んで、それを演算ユニットＡＬＵ（Arithmetic & Logic Unit）1605や、レジスタ1610、1613に転送する。

【００５４】

以下、説明のため、ＡＬＵ1605は8ビットの演算器で、バスライン1606、1607、1608のサイズは16ビットであるものとする。説明のためレジスタは２つのみ考える。またレジスタ1610はソース側、レジスタ1613はディスティネーション側として利用するものとする。レジスタ1610のサイズは16ビットで、HIGH側1609の8ビット と、LOW側1611の8ビットから構成されているものとする。また、演算結果のフラグ、例えば値がゼロであれば1、ゼロでなければ0となるゼロフラグや、桁上がりを示すキャリーフラグ等を格納するレジスタCCR（コンディション･コード･レジスタ）1615があり、ＡＬＵ1605及び各種のバス（1606、1607、1608）に接続されているものとする。

【００５５】

上記のマイクロコンピュータの構成は、簡略化したものであり、一般には、8ビットだけでなく、16ビット、32ビット、64ビットのＣＰＵがあり、レジスタの本数やサイズも異なるものがある。また、ＲＡＭの特定領域をレジスタに割り当

てているものもある。また簡単のため、ＲＯＭやＥＥＰＲＯＭ（ＦＲＡＭ）も示していないが、プログラムは通常このいずれかに保持され、このプログラム中の各命令が図示しないデコーダによってデコードされ、実行されることによってプログラムが実行される。本発明の趣旨は、このような構成の違いには無関係であるので、以下、図２０のマイクロコンピュータの構成を前提に話を進める。

【００５６】

ＡＬＵ1605では、種々の二項演算が行なわれる。論理演算であるAND（論理積）、 OR（論理和）、 XOR（排他的論理和）や、算術演算である＋（和）、－（差）、×（積）、÷（商）は、その代表的なものである。（算術演算は、論理演算回路から構成されている。そのため、ＡＬＵのような回路を「論理回路」と呼ぶことがある。） ２項演算は、通常、以下の二通りの方法で行なわれる。

【００５７】

イミディエート方式： 定数とレジスタの値を演算するもの（ ADD 01、 Rd等）

レジスタ直接方式： レジスタとレジスタの値を演算するもの（ ADD Rs、 Rd等）

本発明は、後者の「レジスタ直接方式」に関するものである。

【００５８】

本発明の趣旨を明瞭、かつ網羅的に説明するために、多桁の数同士の加法を例にとる。ADD.W をワードサイズ（16ビット）のキャリー付きの加法演算を表すニーモニックとする。例えば16*3ビットの数A、Bの加法は、通常、ADD.Wを3回繰り返し用いることによって行われる。この際、繰り上がりに伴いキャリーが発生する。

【００５９】

図２１は、16*3ビットの数Aと、同じく16*3ビットの数Bの和を計算する過程を表現したものである。Aは3つのワード（16ビットブロック）から構成される。これを下位から順にA[0]、 A[1]、 A[2]とする。同様に、BについてもB[0]、 B[1]、 B[2]とする。ADD.W命令によって、A[0]とB[0]の和を計算し、和をC[0]と、キャリー（1712）として出力する。キャリーは、CCR（コンディション・コード・レ

ジスタ）と呼ばれるレジスタに格納される。） 次にビットブロックA[1]、 B[1]の和および先の計算におけるキャリー（1712）の和をADD.Wによって計算し、この和をC[1]と、キャリー（1716）として求める。次にビットブロックA[2]、 B[2]の和および先の計算におけるキャリー（1716）の和をADD.Wによって計算し、この和をC[2]と、キャリー（1718）として求める。これを「キャリー（1718）C[2]C[1]C[0]」の順に上位から並べたものが、AとBの和Cとなる。

【００６０】

この計算を本発明の方法を用いて実装する方法を述べる。上記の説明にあるように、ワードサイズを超えるサイズの数A、Bの和を計算するには、A及びBをワード単位に分割して演算ADD.Wをワードサイズのブロック数と同じ回数だけ繰り返し、その際、各ブロック部の計算で生じたキャリーを順次繰り上げて処理することによって目的の和を求めることができる。原理的にはいくらでも大きな桁の数の和が計算できる。

【００６１】

以下、A = (A[n-1]A[n-2]...A[1]A[0])、 B = (B[n-1]B[n-2]...B[1]B[0])（各A[j]、 B[j]は、16ビットブロック）のように表示する。図２２は、このA、Bの和を計算する手順を示したものである。まず、A、Bを受信する（ステップ1801）。ここで受信と言っているのは、I/Oポートから信号を受信するという意味の他、計算途中で、A、Bが確定したということをも意味するものとする。次にカウンタjを0に初期化する（ステツプ1802）。条件分岐処理（ステップ1803）は、カウンタがnであるかどうかを判定するものであり、j = nのときは、全てのビットブロックの処理が終了したことを意味しているので終了し、そうでないときはステップ1804に進む。ステップ1804においては、1ビットの乱数vを発生する。乱数発生後、ステップ1805の条件分岐処理にてvが0であるか1であるか判定し、0であればA[j]をソースレジスタRsに転送し（ステップ1806）、その後、B[j]をディスティネーションレジスタRdに転送する（ステップ1807）。その後、ADD.W Rs、 Rdを実行する（ステップ1810）。ステップ1810においては、Rsの値とRdの値の和を計算し、その値をRdに転送する（ステップ1811）。繰り上がりがあれば、キャリーとして保持し、次回のADD.W処理の際に1を加える処理を行なう。この部分は

、ADD.Wという命令が処理するものであるので、処理のフローには書かれていないが、通常、アセンブラ言語のプログラマが、キャリの処理を行なうことはあまりない。次に、Rdの内容をＲＡＭのCの所定位置C[j]に格納する（ステップ1811）。次にjをインクリメントして（ステツプ1812）、再びステップ1803の条件分岐処理に戻る。逆にvが1であれば、B[j]をディスティネーションレジスタRdに転送し（ステップ1808）、その後、A[j]をソースレジスタRsに転送する（ステップ1809）。その後、ADD.W Rs、 Rdを実行する（ステップ1810）。ステップ1810においては、Rsの値とRdの値の和を計算し、その値をRdに転送する。繰り上がりがあればキャリーとして保持し、次回のADD.W処理の際に1を加える処理を行なう。次に、Rdの内容をＲＡＭのCの所定位置C[j]に格納する（ステップ1811）。次にjをインクリメントして（ステップ1812）、再びステップ1803の条件分岐処理に戻る。

【００６２】

この処理においては、乱数ｖを発生させ、その値によって、A[j]をRsに転送してからB[j]をRdに転送するか、その逆にB[j]をRdに転送してからA[j]をRsに転送するかを切り換えている。この切り換えによってＩＣチップの消費電力の波形が変化し、特に波形を平均化してノイズを除去する処理（通常のオシロスコープでは、この方法でノイズを除去する）を行なってデータの違いを読み取ろうとすると、波形がＡとＢの平均としてしか観測されず、Ａ、Ｂ個別の内容が推定できなくなる。個別に例を示すことはしないが、本実施例を16ビットブロック単独の和に制限することは容易なことである。また本実施例では乱数ｖを用いて条件分岐処理をスイッチしているが、これを線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係であることはいうまでもない。

【００６３】

本実施例が和ではなく、積の場合にどうなるかを示す。積の場合も本質は全く同様である。ここでは、MULTIという命令を想定する。MULTI RsL、 RdLは、RsのLOW側の8ビットと、RdのLOW側の8ビットとを乗じて、Rd（16ビット）に積を格納するものとする。MULTIを用いて、16ビットの数A、 Bの積を計算する処理を図

２３に示す。ここでは、A = (A[1]A[0])、 B = (B[1]B[0])、A*B＝Cを(C[3]C[2]C[1]C[0]) （ここで、A[j]、 B[j]、 C[j]は8ビットのブロック）と表示する。まずA[0]とB[0]の積を積演算MULTIにて計算する。積は最大で16ビットの数になる。この値は、Rdに格納されている。これを上位からT[0] [1]、T[0] [0]とする。T[0] [0]は、そのままC[0]とする。次に、A[1]と、B[0]の積を計算し上位からT[1] [1]、T[1] [0]とする。次に、B[1]とA[0]の積を計算し、上位からT[2] [1]、T[2] [0]とする。T[0] [1]と、T[1] [0]と、T[2] [0]の和をADD.B（8ビットのキャリー付和演算）を二回用いて計算し、キャリーとC[1]を得る。T[1] [1]と、T[2] [1]と、T[3] [0]の和をADD.B（8ビットのキャリー付和演算）を二回用いて計算し、キャリーとC[2]を得る。このキャリーとT[3] [1]をADD.Bで加えてC[3]を得る。最後ではキャリーは生じない。

【００６４】

ここでは、A = (A[n-1]A[n-2]...A[1]A[0])、 B = (B[n-1]B[n-2]...B[1]B[0])（各A[j]、 B[j]は、8ビットブロック）のように表示する。また、ＲＡＭ上に一時記憶領域TMP1、TMP2という8*nビットの互いに重複のない領域があるものとする。多ビットの和の処理部は、図２１の説明に任せ、以下では特に説明しないものとする。

【００６５】

図２４及び図２５は、A、Bの積を計算する手順を示したものである。まず、A、Bを受信する（ステップ2001）。ここで受信と言っているのは、I/Oポートから信号を受信するという意味の他、計算途中でA、Bが確定したということをも意味するものとする。次にカウンタi及びjを0に初期化する（ステップ2002）。条件分岐処理（ステップ2003）は、カウンタjがnであるかどうかを判定するものであり、j = nのときは、全てのビットブロックの処理が終了したことを意味しているので終了し、そうでないときは、ステップ2004に進む。ステップ2004においては、1ビットの乱数vを発生する。乱数発生後、条件分岐処理（ステップ2005）にて、vが0であるか1であるか判定し、0であればA[i]をソースレジスタRsLに転送し（ステップ2006）、その後、B[j]をディスティネーションレジスタRdLに転送する（ステップ2007）。その後、MULTI RsL、 Rdを実行し（ステップ2010）、部

分積の値が格納されているディスティネーションレジスタRdの値をＲＡＭ上の一次記憶領域TMP1に転送する（ステップ2011）。次にカウンタiを1だけインクリメントし（ステップ2012）、条件分岐処理（ステップ2013）に進む。条件分岐処理（ステップ2013）は、カウンタiがnであるかどうか判定し、i = nであれば、jを1だけインクリメントし、iを0に初期化して（ステップ2025）、条件分岐処理（ステップ2003）に戻り、先に述べた操作を行なう。条件分岐処理（ステップ2013）においてiがnでなければ、1ビットの乱数vを発生し（ステップ2015）、v = 0であれば、A[i]をRsLに転送（ステップ2016）した後、B[j]をRdLに転送する（ステップ2017）。逆にv = 1の場合は、B[j]をRdLに転送（ステップ2018）した後、A[i]をRsLに転送する（ステップ2019）。ステップ2017かステップ2019のいずれかの処理が終了した後、MULTI RsL、 Rdを実行し（ステップ2020）、RdをTMP2に転送し（ステップ2021）、TMP2の値を左に8ビットシフトし（ステップ2022）、TMP1とTMP2の和（この操作に関しては、図２１の説明参照）を計算してTMP1に転送し（ステップ2023）、iを1インクリメントして（ステップ2024）、条件分岐処理（ステップ2013）に戻る。

【００６６】

この処理においては、乱数ｖを発生させ、その値によってA[j]をRsに転送してからB[j]をRdに転送するか、その逆にB[j]をRdに転送してからA[j]をRsに転送するかを切り換えている。この切り換えによってＩＣチップの消費電力の波形が変化し、特に波形を平均化してノイズを除去する処理（通常のオシロスコープでは、この方法でノイズを除去する）を行なってデータの違いを読み取ろうとすると、波形がＡとＢの平均としてしか観測されず、Ａ、Ｂ個別の内容が推定できなくなる。個別に例を示すことはしないが、本実施例を8ビットブロック単独の和に制限することは容易なことである。また本実施例では乱数ｖを用いて条件分岐処理をスイッチしているが、これを線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係であることはいうまでもない。

上記の実施例は、AをソースレジスタRsに、BをディスティネーションレジスタRdに転送する順序を変えるものであった。この類似として、AをソースレジスタRs

に転送して後、Bをディスティネーションレジスタ Rdに転送するか、Aをディスティネーションレジスタ Rdに転送して後、Bをソースレジスタ Rsに転送するかをランダムにスイッチすることが考えられる。この実施例を和の場合に限って示す。上記の実施例に僅かな変更を加えるに過ぎないので、積の場合等、容易に類推できるものについては省略する。ここでは、以下、A = (A[n-1]A[n-2]...A[1]A[0])、 B = (B[n-1]B[n-2]...B[1]B[0])（各A[j]、 B[j]は、16ビットブロック）のように表示する。 図２６は、このA、Bの和を計算する手順を示したものである。まず、A、Bを受信する（ステップ2101）。ここで、受信と言っているのは、I/Oポートから信号を受信するという意味の他、計算途中で、A、Bが確定したということをも意味するものとする。次に、カウンタjを0に初期化する（ステップ2102）。条件分岐処理（ステップ2103）は、カウンタがnであるかどうかを判定するものであり、j = nのときは、全てのビットブロックの処理が終了したことを意味しているので終了し、そうでないときは、ステップ2104に進む。ステップ2104においては、1ビットの乱数vを発生する。乱数発生後、条件分岐処理（ステップ2105）にて、vが0であるか1であるか判定し、0であればA[j]をソースレジスタ Rsに転送し（ステップ2106）、その後、B[j]をディスティネーションレジスタ Rdに転送する（ステップ2107）。その後、ADD.W Rs、 Rdを実行する（ステップ2110）。ステップ2110においては、Rsの値とRdの値の和を計算し、その値をRdに転送する。繰り上がりがあれば、キャリーとして保持し、次回のADD.W処理の際に1を加える処理を行なう。この部分は、ADD.Wという命令が処理するものであるので、処理のフローには書かれていないが、通常、アセンブラ言語のプログラマが、キャリの処理を行なうことはあまりない。次に、Rdの内容をＲＡＭのCの所定位置C[j]に格納する（ステップ2111）。次にjをインクリメントして（ステップ2112）、再び条件分岐処理（ステップ2103）に戻る。逆にvが1であれば、A[j]をディスティネーションレジスタ Rdに転送し（ステップ2108）、その後、B[j]をソースレジスタ Rsに転送する（ステップ2109）。その後、ADD.W Rs、 Rdを実行する（ステップ2110）。ステップ2110においては、Rsの値とRdの値の和を計算し、その値をRdに転送する。繰り上がりがあれば、キャリーとして保持し、次回のADD.W処理の際に1を加える処理を行なう。次に、Rdの内容をRAMのCの所定位置C[j]

に格納する（ステップ2111）。次にjをインクリメントして（ステップ2112）、再び条件分岐処理（ステップ2103）に戻る。乱数による処理の切り換えによってＩＣチップの消費電力の波形が変化し、アタッカーは撹乱される。但し、この方式では、転送レジスタを切り換えているだけなので、図２２に示す実施例のように、データの平均化の効果は期待できない。マイクロコンピュータによっては、RsとRdに転送する際の消費電流に大きな違いがある場合があるが、そのような場合には、本実施例の処理が有効であると考えることができる。個別に例を示すことはしないが、本実施例を、16ビットブロック単独の和に制限することは容易なことである。また、本実施例では、乱数ｖを用いて条件分岐処理をスイッチしているが、これを、線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係であることはいうまでもない。

【００６７】

また、本実施例と図２２に示す実施例を混合して用いるなどすれば、高い撹乱効果が期待できる。

【００６８】

次にさらに他の実施例について説明する。上記の実施例では、ビットブロックに区切って（例えば、8ビットや16ビットに区切って）和や積のような算術演算を計算する場合に繰り上がり（キャリー。引き算の場合はボロー）が生ずるので、各ブロック毎に独立に演算を実行することができない。一方、排他的論理和EXOR、論理積AND、論理和OR等の論理演算は、ビット毎の処理を行なうものであり、繰り上がり等の他のビットブロックの処理に影響するような処理が生ずることはない。このような場合には、上記実施例よりも大きな撹乱処理を行うことができる。

【００６９】

図２７は、このような実施例を特にA = (A[n-1]A[n-2]...A[1]A[0])、 B = (B[n-1]B[n-2]...B[1]B[0])（A[j]、 B[j]は8ビットブロック）に対し、A EXOR Bを計算する場合に適用したものである。ANDや、ORにした場合も、同様であるので、省略する。（参考までに、論理演算EXOR、 AND、 ORの演算表を、図２８、

２９、３０に示しておく。） ここでは、A EXOR Bの演算結果をC = (C[n-1]C[n-2]...C[1]C[0])のように表示する。 また、図２７において、PERM[v]は、確率変数vに従って決まる置換で、PERM[v][j] は、その第j成分(j=0,1,...n-1)であるものとする。具体的には、例えば、n = 3の場合なら、PERM[v]は、6通りの置換

PERM[0] (0 1 2) → (0 1 2)

PERM[1] (0 1 2) → (0 2 1)

PERM[2] (0 1 2) → (1 0 2)

PERM[3] (0 1 2) → (1 2 0)

PERM[4] (0 1 2) → (2 0 1)

PERM[5] (0 1 2) → (2 1 0)

のどれかである。表示法を確認するために、例を挙げると、例えば、PERM[3][1] = 2（PERM[3]の第１成分）、 PERM[5][2] = 0（PERM[5]の第2成分）である。この置換は、RAMに配置する。

【００７０】

本実施例においては、確率変数vは、0から n! -1(n! = n*(n-1)*...*2*1)を値に取る一様分布に従うものとする。

【００７１】

以上を前提に図２７の説明に入る。まず、演算に用いるA、 Bを受信する（ステップ2501）。次にカウンタjを0に設定し（ステップ2502）、乱数vを発生する（vは上記のものである）（ステップ2503）。次に条件分岐処理（ステップ2504）に入る。条件分岐処理（ステップ2504）では、カウンタjがnであるかどうか判定し、j = nであれば、全てのビットブロックを処理したことになるので、演算処理を終了する。もし、jがnでなければ、k = PERM[v][j]とし（ステップ2505）、A[k]をRsLに転送し（ステップ2506）、B[k]をRdLに転送し（ステップ2507）、排他的論理和EXOR RsL RdLを実行する（ステップ2508）。この答が格納されているRdLの値をC[k]に転送し（ステップ2509）、カウンタを１インクリメントし（ステップ2510）、再び条件分岐処理（ステップ2504）に戻る。個別に例を示すことはしないが、本実施例を、AND、OR等、他の論理演算に置き換えることは容易

である。実際、図２７のステップ2508の演算をAND RsL、 RdLやOR RsL、 RdLとすることで目的は達せられる。また、本実施例では、乱数ｖを用いて条件分岐処理をスイッチしているが、これを、線形合同法等を用いた疑似乱数や、カオス数列や、あらかじめ定められたビット列に変えることは容易なことであり、本発明の本質とは無関係であることはいうまでもない。

【００７２】

また、本実施例において、図２２や図２６の実施例を、ステップ2506の処理とステップ2507の処理の順序の変更に利用することは、本発明を単独に用いることよりも、高い効果の期待できるものである。

【００７３】

【発明の効果】

本発明によれば、ＩＣカードチップなどの情報処理装置において、ＣＲＴ（中国人剰余定理）を用いたべき乗剰余計算を実行する際に、ＣＲＴのために算術的関係を持って分割された指数に対するべき乗剰余処理の途中計算を本来の順序とは別の順序で処理することにより、消費電流の波形から処理や暗号鍵の推測を行うことが困難になる。また、和や積を計算する際の処理順序が本来の順序とは別の順序で処理することにより、消費電流の波形から処理や暗号鍵の推測を行なうことが困難になる。

【図面の簡単な説明】

【図１】

ＩＣカードの概観及び端子を示す図である。

【図２】

マイクロコンピュータの構成図である。

【図３】

消費電流の波形を示す図である。

【図４】

ＣＲＴ（中国人剰余定理）を用いたべき乗剰余計算の手順を示す図である。

【図５】

2ビットアディション・チェイン方式によるべき乗剰余計算アルゴリズムを示す

図である。

【図６】

2ビットアディション･チェイン方式によるべき乗剰余計算に関する実施例を示す図である。

【図７】

２ビットアディション･チェイン方式によるべき乗剰余計算に関する実施例を示す図（続き）である。

【図８】

実施例の ２ビット毎の乗算剰余処理ルーチン（モンゴメリ法）を示す図である。

【図９】

２ビットスライディング･ウインドウ方式によるべき乗剰余計算に関する実施例を示す図である。

【図１０】

２ビットスライディング･ウインドウ方式によるべき乗剰余計算に関する実施例を示す図（続き）である。

【図１１】

実施例のスライディングウインドウ方式による乗算剰余処理ルーチン（モンゴメリ法）を示す図である。

【図１２】

「ｘの一部のビット列」の例（１）を示す図である。

【図１３】

「ｘの一部のビット列」の例（２）を示す図である。

【図１４】

「ｘの一部のビット列」の例（３）を示す図である。

【図１５】

「ｘの一部のビット列」の例（４）を示す図である。

【図１６】

xp、 xqについて一部のビット列の例を示す図である。

【図１７】

実施例の一部のビット列について本発明を適用する処理手順を示す図である。

【図１８】

実施例の一部のビット列について本発明を適用する処理手順を示す図（続き）である。

【図１９】

実施例の一部のビット列について本発明を適用する処理手順を示す図（続き）である。

【図２０】

マイクロコンピュータの中央演算装置及びＲＡＭの構成例を示す図である。

【図２１】

ビットブロックに分割された二数A、Bの和の計算法の例を示す図である。

【図２２】

実施例の和演算の処理手順を示す図である。

【図２３】

ビットブロックに分割された二数A、Bの積の計算法の例を示す図である。

【図２４】

実施例の積演算の処理手順を示す図である。

【図２５】

実施例の積演算の処理手順を示す図（続き）である。

【図２６】

他の実施例の和演算の処理手順を示す図である。

【図２７】

他の実施例の排他的論理和の処理手順を示す図である。

【図２８】

排他的論理和EXORの演算表を示す図である。

【図２９】

論理積ANDの演算表を示す図である。

【図３０】

論理和ORの演算表を示す図である。

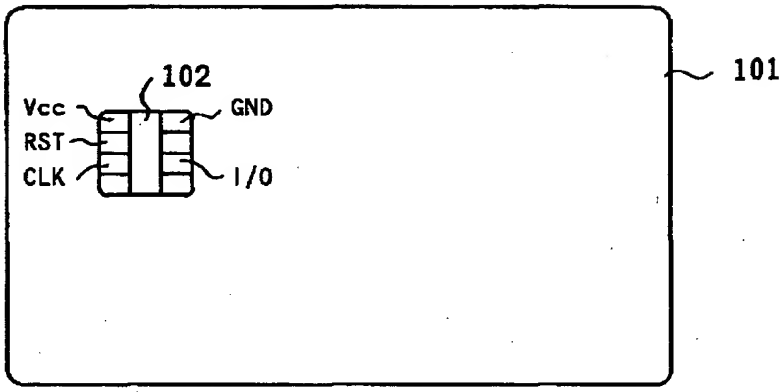
【符号の説明】

606：ステップ（乱数生成）、607：ステップ（乱数値による処理選択）、608：ステップ（Cpの計算）、609：ステップ（Cqの計算）、1804,2004,2014,2104,2503：ステップ（乱数生成）、1805,2005,2015,2105：ステップ（乱数値による処理選択）

【書類名】　図面

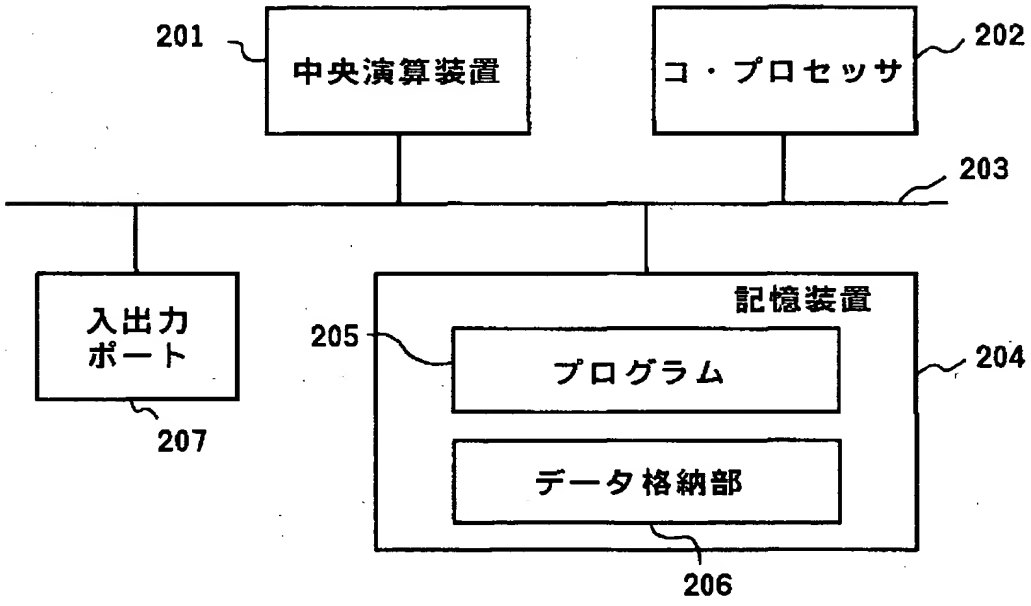
【図１】

図　1



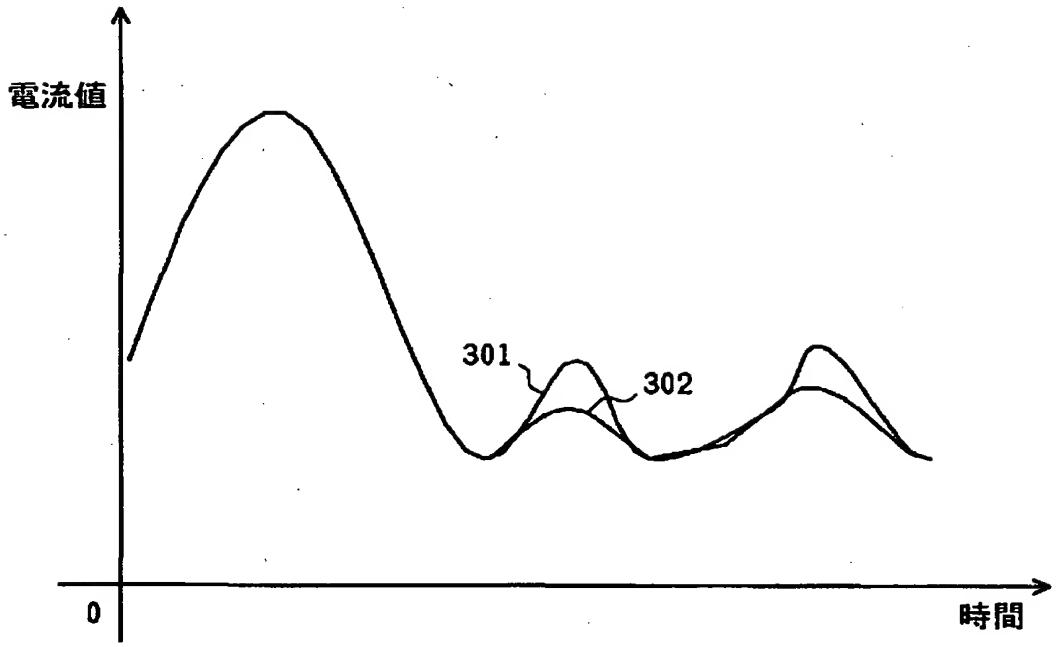
【図２】

図　2



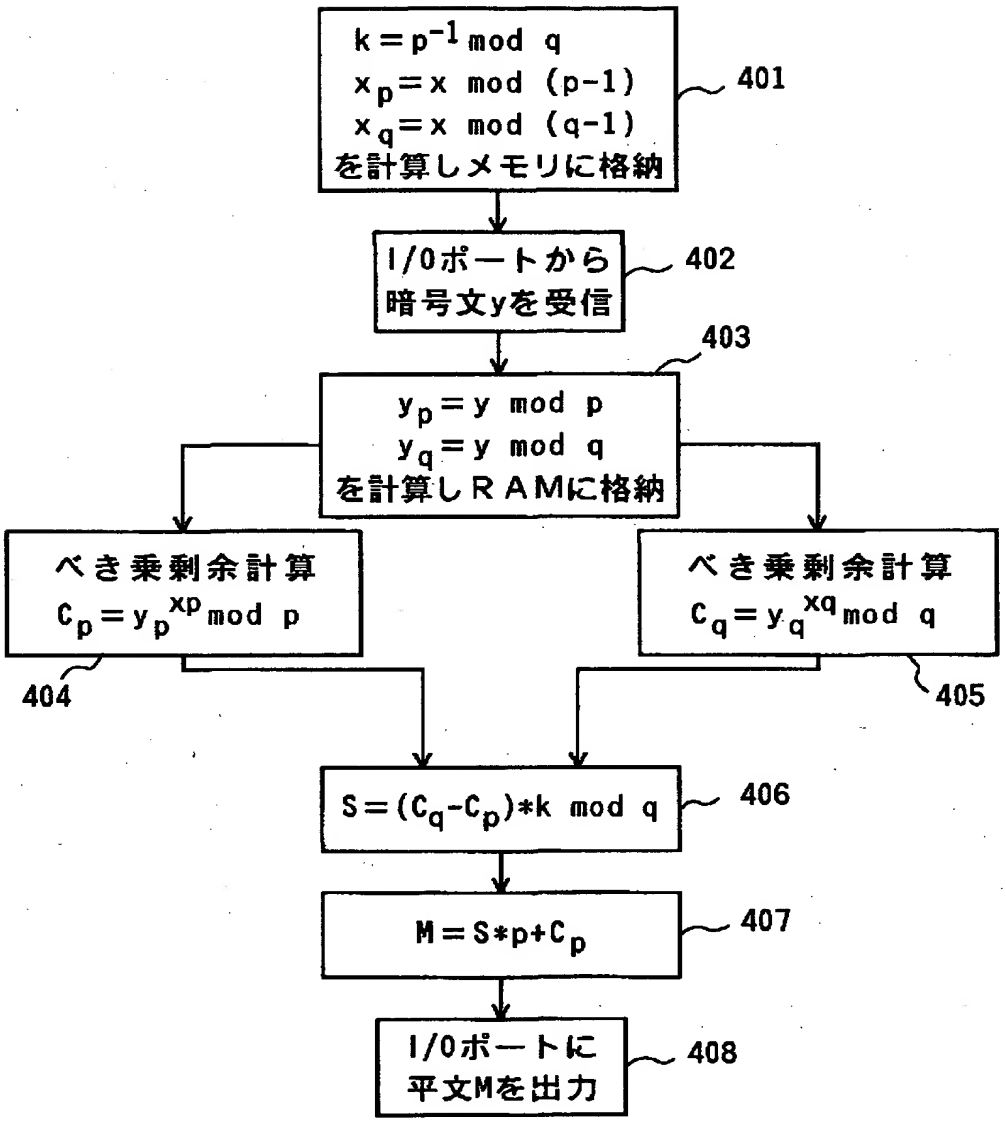
【図３】

図　3



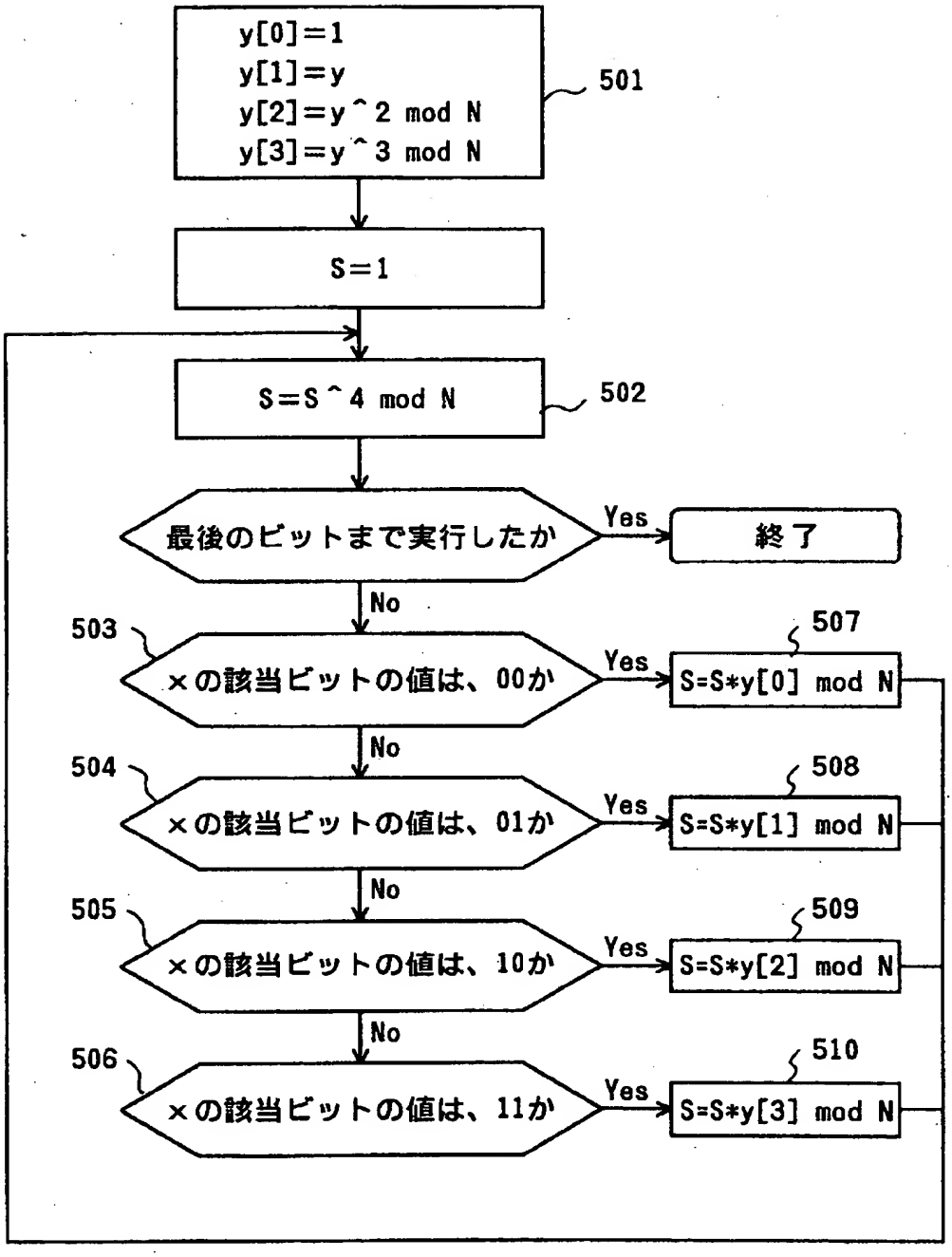
【図４】

図　4


k＝p^-1 mod q
xp＝x mod (p-1)
xq＝x mod (q-1)
を計算しメモリに格納
401
I/Oポートから
暗号文yを受信
402
403
yp＝y mod p
yq＝y mod q
を計算しＲＡＭに格納
べき乗剰余計算
Cp＝yp^xp mod p
べき乗剰余計算
Cq＝yq^xq mod q
404
405
S＝(Cq-Cp)*k mod q
406
M＝S*p+Cp
407
I/Oポートに
平文Mを出力
408

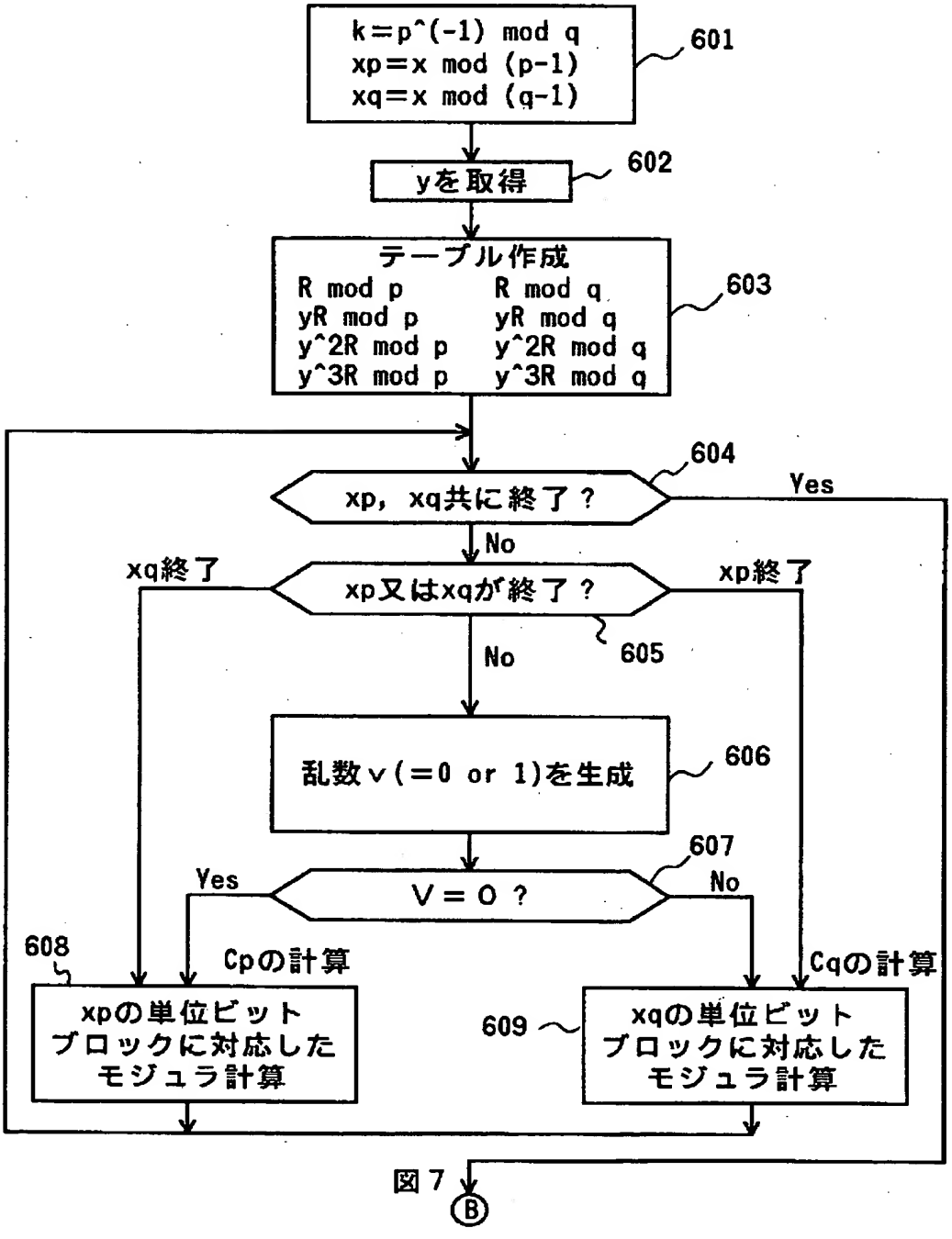
【図５】

図　5


y[0]＝1
y[1]＝y
y[2]＝y＾2 mod N
y[3]＝y＾3 mod N
501
S＝1
S＝S＾4 mod N
502
最後のビットまで実行したか
Yes
終了
No
503
×の該当ビットの値は、00か
Yes
507
S=S*y[0] mod N
No
504
×の該当ビットの値は、01か
Yes
508
S=S*y[1] mod N
No
505
×の該当ビットの値は、10か
Yes
509
S=S*y[2] mod N
No
506
×の該当ビットの値は、11か
Yes
510
S=S*y[3] mod N

【図６】

図　6


k＝p^(-1) mod q
xp＝x mod (p-1)
xq＝x mod (q-1)
601
yを取得
602
テーブル作成
R mod p
R mod q
yR mod p
yR mod q
y^2R mod p
y^2R mod q
y^3R mod p
y^3R mod q
603
604
xp，xq共に終了？
Yes
No
xq終了
xp又はxqが終了？
xp終了
605
No
乱数v(＝0 or 1)を生成
606
607
Yes
V＝0？
No
608
Cpの計算
Cqの計算
xpの単位ビット
ブロックに対応した
モジュラ計算
609
xqの単位ビット
ブロックに対応した
モジュラ計算
図7
Ⓑ

【図７】

図　７

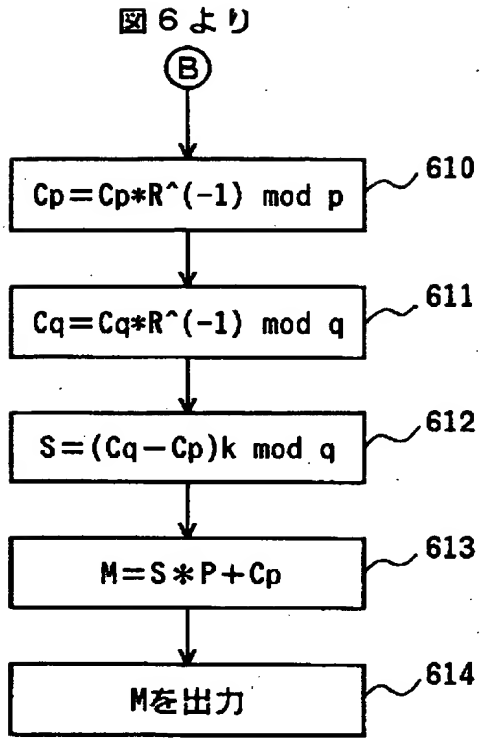
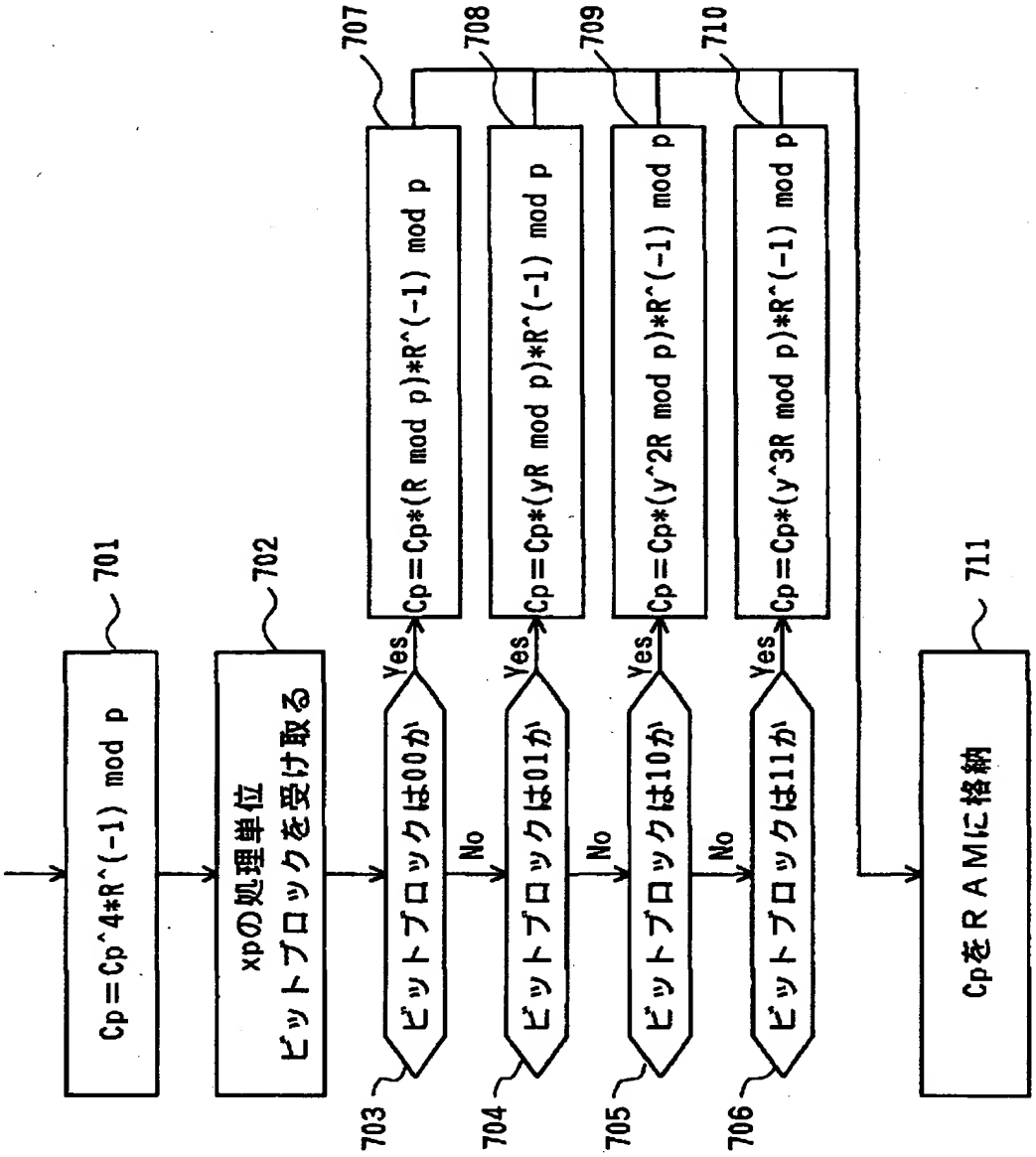


図６より
B
Cp＝Cp*R^(-1) mod p
610
Cq＝Cq*R^(-1) mod q
611
S＝(Cq－Cp)k mod q
612
M＝S＊P＋Cp
613
Mを出力
614

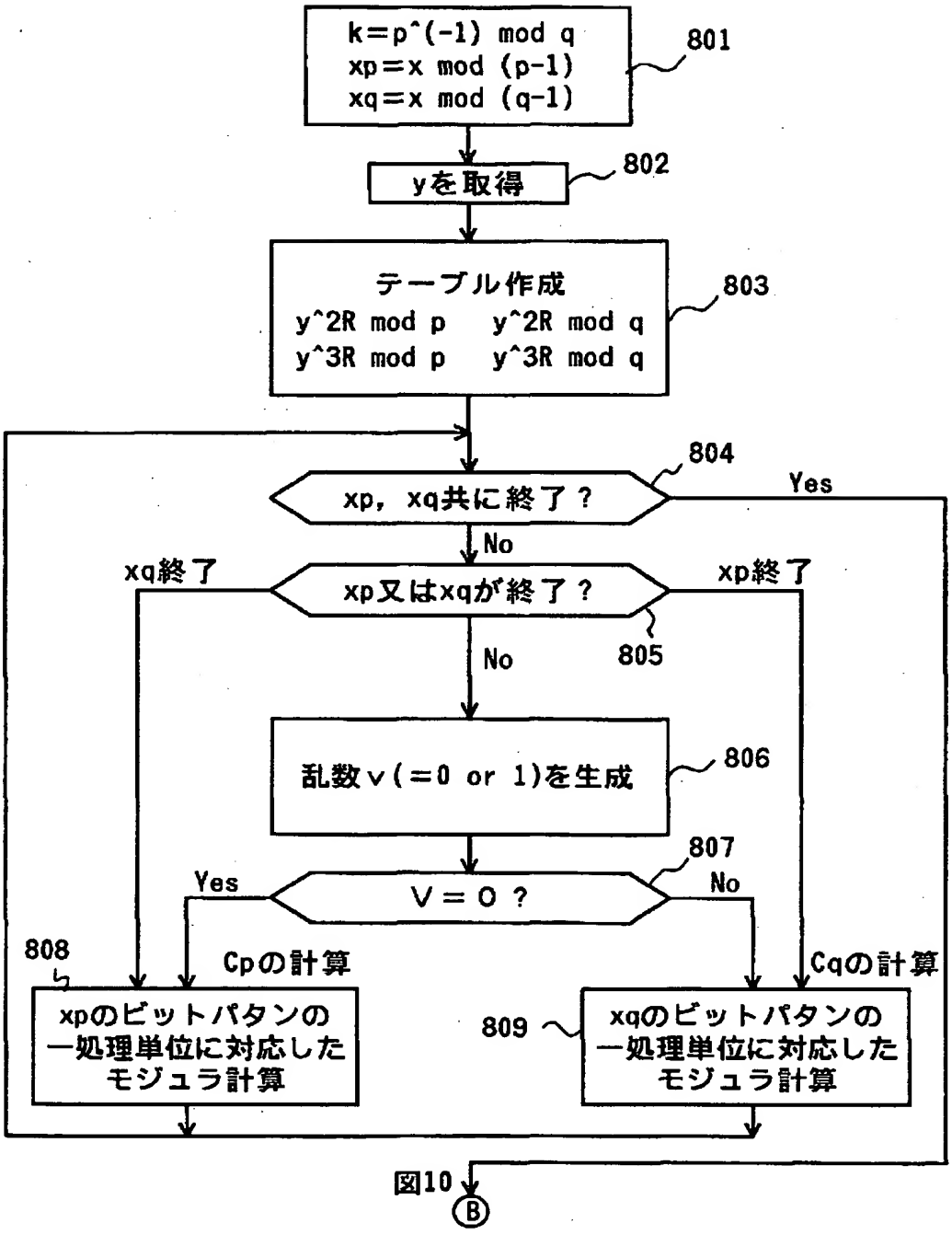
【図８】

図　８


701
Cp=Cp^4*R^(-1) mod p
702
xpの処理単位
ビットブロックを受け取る
703
ビットブロックは00か
Yes
No
704
ビットブロックは01か
Yes
No
705
ビットブロックは10か
Yes
No
706
ビットブロックは11か
Yes
No
707
Cp=Cp*(R mod p)*R^(-1) mod p
708
Cp=Cp*(yR mod p)*R^(-1) mod p
709
Cp=Cp*(y^2R mod p)*R^(-1) mod p
710
Cp=Cp*(y^3R mod p)*R^(-1) mod p
711
CpをＲＡＭに格納

【図９】

図　9


k=p^(-1) mod q
xp=x mod (p-1)
xq=x mod (q-1)
801
yを取得
802
テーブル作成
y^2R mod p　y^2R mod q
y^3R mod p　y^3R mod q
803
804
xp，xq共に終了？
Yes
No
xq終了
xp又はxqが終了？
xp終了
805
No
乱数v(=0 or 1)を生成
806
807
Yes
V＝０？
No
808
Cpの計算
Cqの計算
xpのビットパタンの一処理単位に対応したモジュラ計算
809
xqのビットパタンの一処理単位に対応したモジュラ計算
図10
Ⓑ

【図１０】

図 10

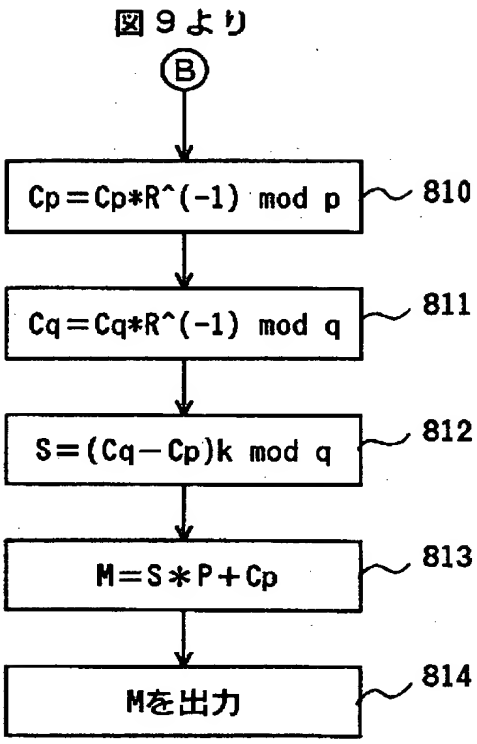
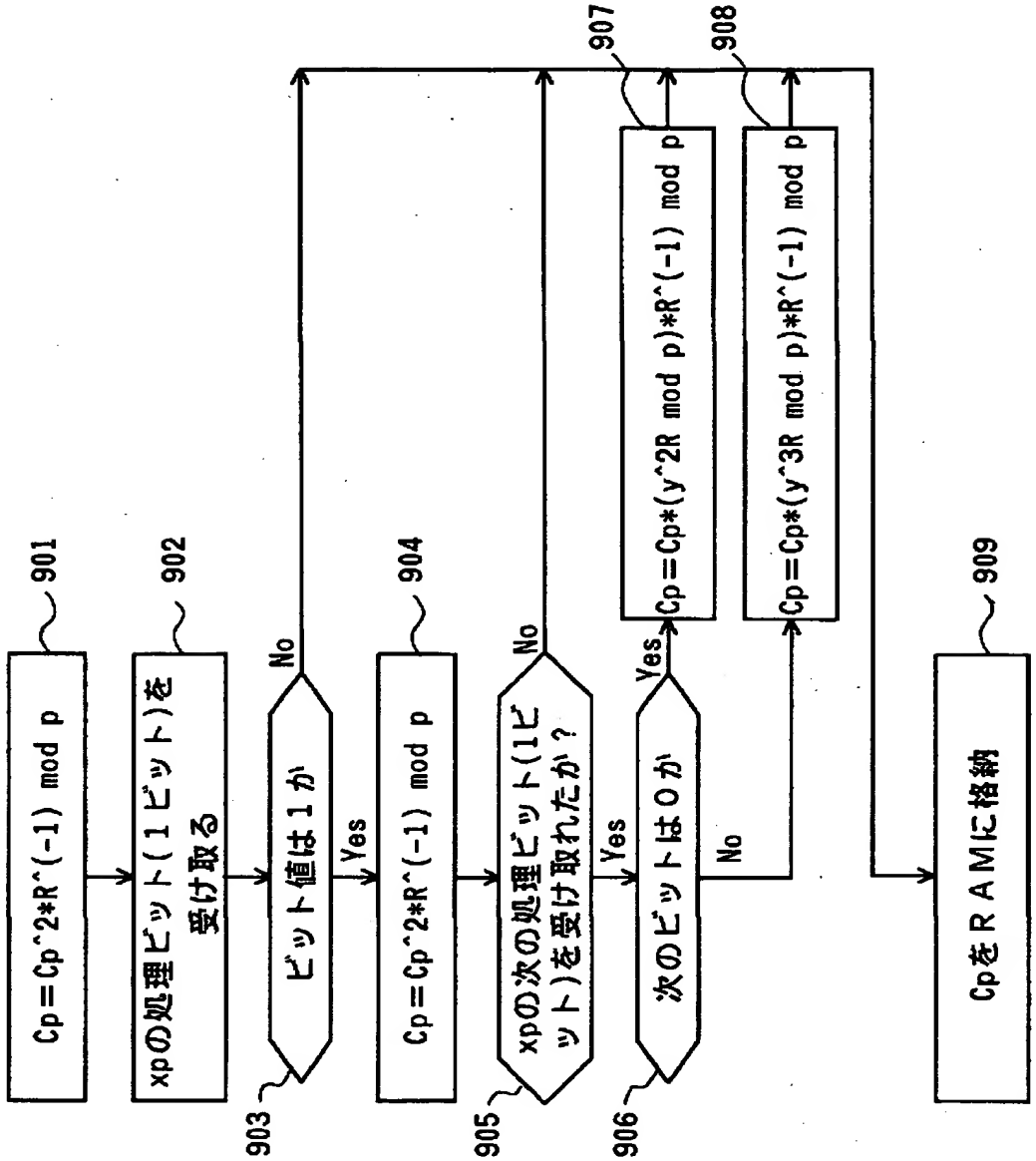


図９より
B
Cp＝Cp*R^(-1) mod p
810
Cq＝Cq*R^(-1) mod q
811
S＝(Cq－Cp)k mod q
812
M＝S＊P＋Cp
813
Mを出力
814

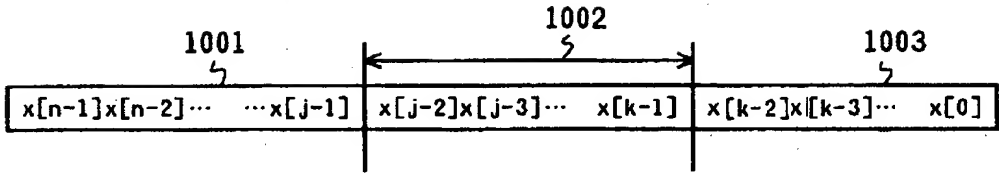
【図１１】

図 11


901
Cp=Cp^2*R^(-1) mod p
902
xpの処理ビット(１ビット)を受け取る
903
ビット値は１か
No
Yes
904
Cp=Cp^2*R^(-1) mod p
905
xpの次の処理ビット(1ビット)を受け取れたか？
No
Yes
906
次のビットは０か
Yes
No
907
Cp=Cp*(y^2R mod p)*R^(-1) mod p
908
Cp=Cp*(y^3R mod p)*R^(-1) mod p
909
CpをＲＡＭに格納

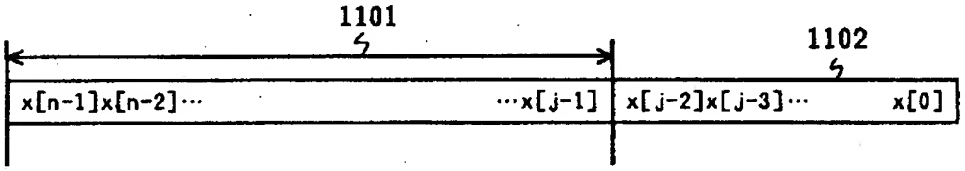
【図１２】

図　12



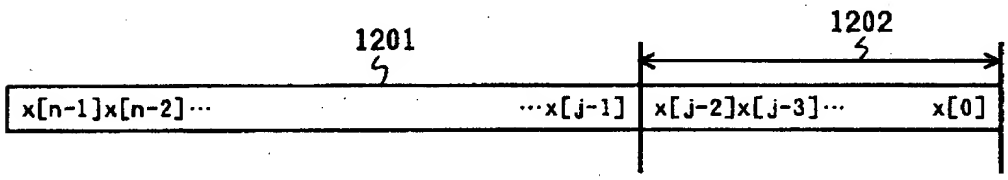
【図１３】

図　13



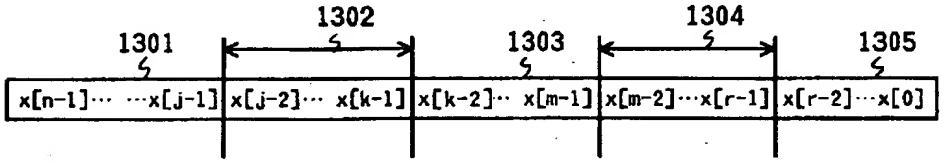
【図１４】

図　14



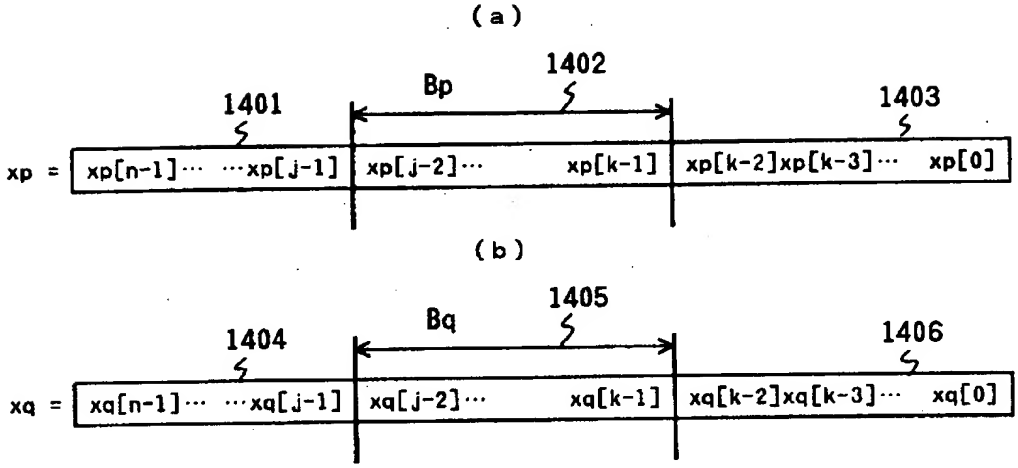
【図１５】

図　15



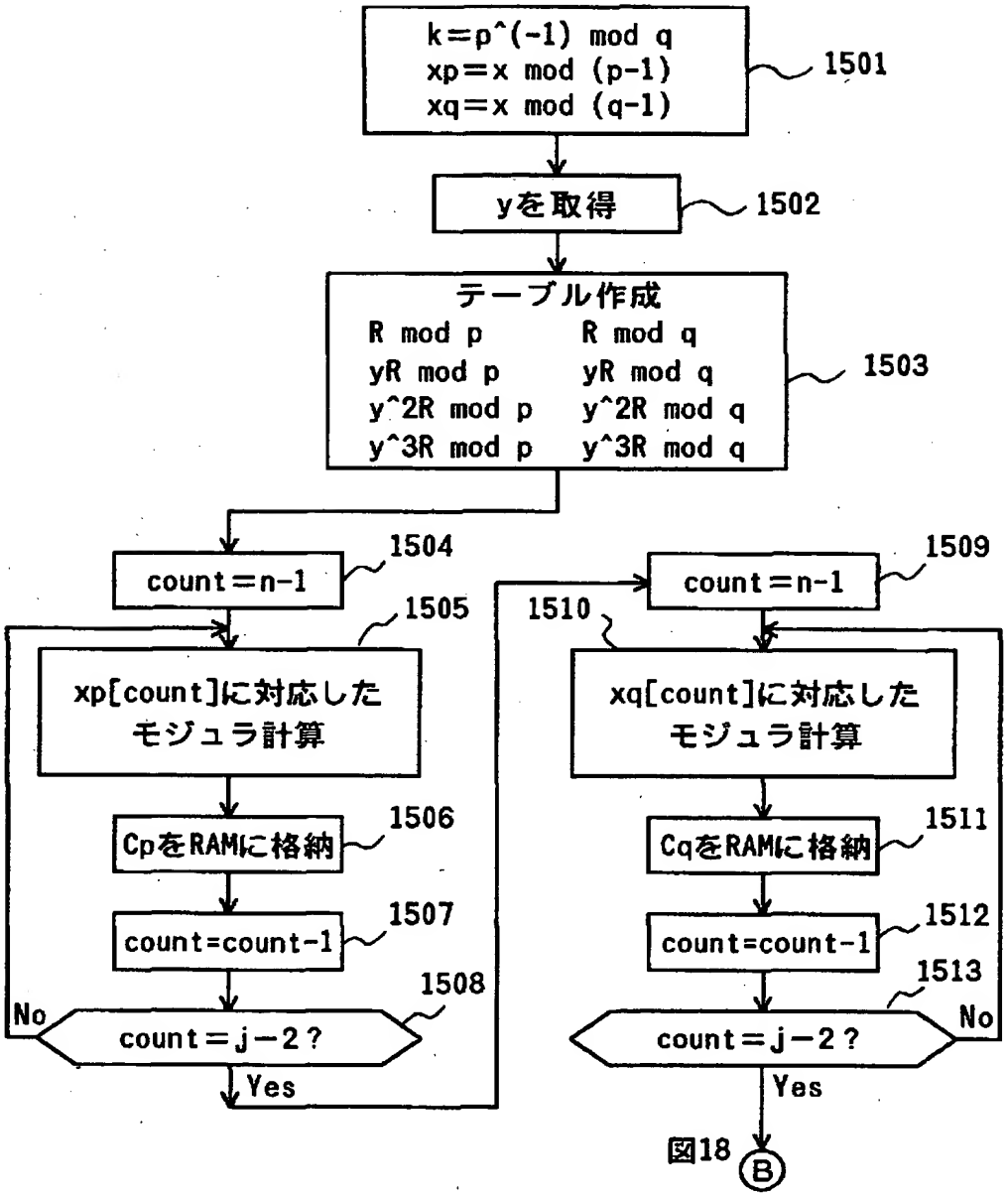
【図１６】

図 16


(a)
1401
Bp
1402
1403
xp =
xp[n-1]… …xp[j-1]
xp[j-2]… xp[k-1]
xp[k-2]xp[k-3]… xp[0]
(b)
1404
Bq
1405
1406
xq =
xq[n-1]… …xq[j-1]
xq[j-2]… xq[k-1]
xq[k-2]xq[k-3]… xq[0]

【図１７】

図　17


k=p^(-1) mod q
xp=x mod (p-1)
xq=x mod (q-1)
1501
yを取得
1502
テーブル作成
R mod p
R mod q
yR mod p
yR mod q
y^2R mod p
y^2R mod q
y^3R mod p
y^3R mod q
1503
count=n-1
1504
1505
xp[count]に対応した
モジュラ計算
CpをRAMに格納
1506
count=count-1
1507
1508
No
count=j-2?
Yes
count=n-1
1509
1510
xq[count]に対応した
モジュラ計算
CqをRAMに格納
1511
count=count-1
1512
1513
No
count=j-2?
Yes
図18
B

【図１８】

図　18

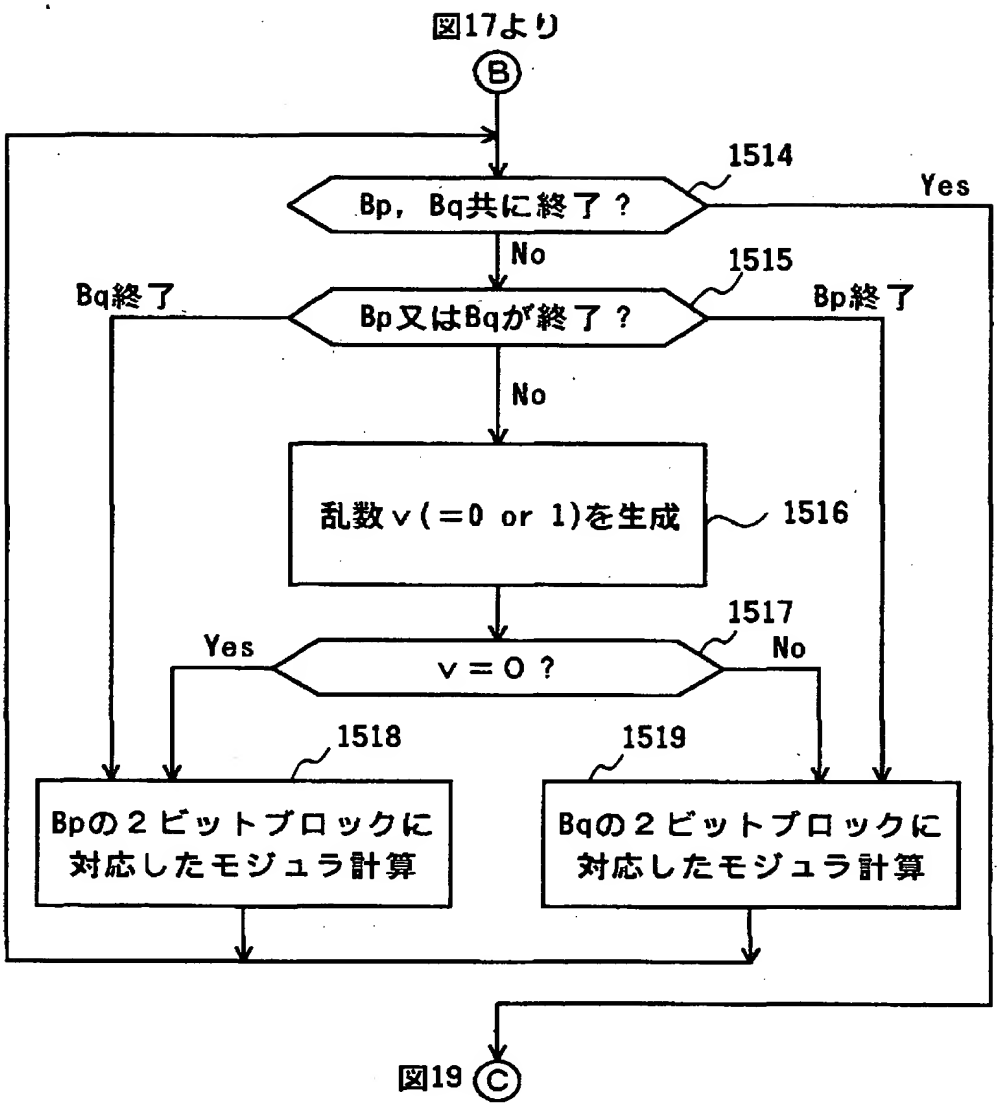


図17より
B
1514
Bp，Bq共に終了？
Yes
No
1515
Bq終了
Bp又はBqが終了？
Bp終了
No
乱数v(＝0 or 1)を生成
1516
1517
Yes
v＝0？
No
1518
1519
Bpの２ビットブロックに
対応したモジュラ計算
Bqの２ビットブロックに
対応したモジュラ計算
図19
C

【図１９】

図　19

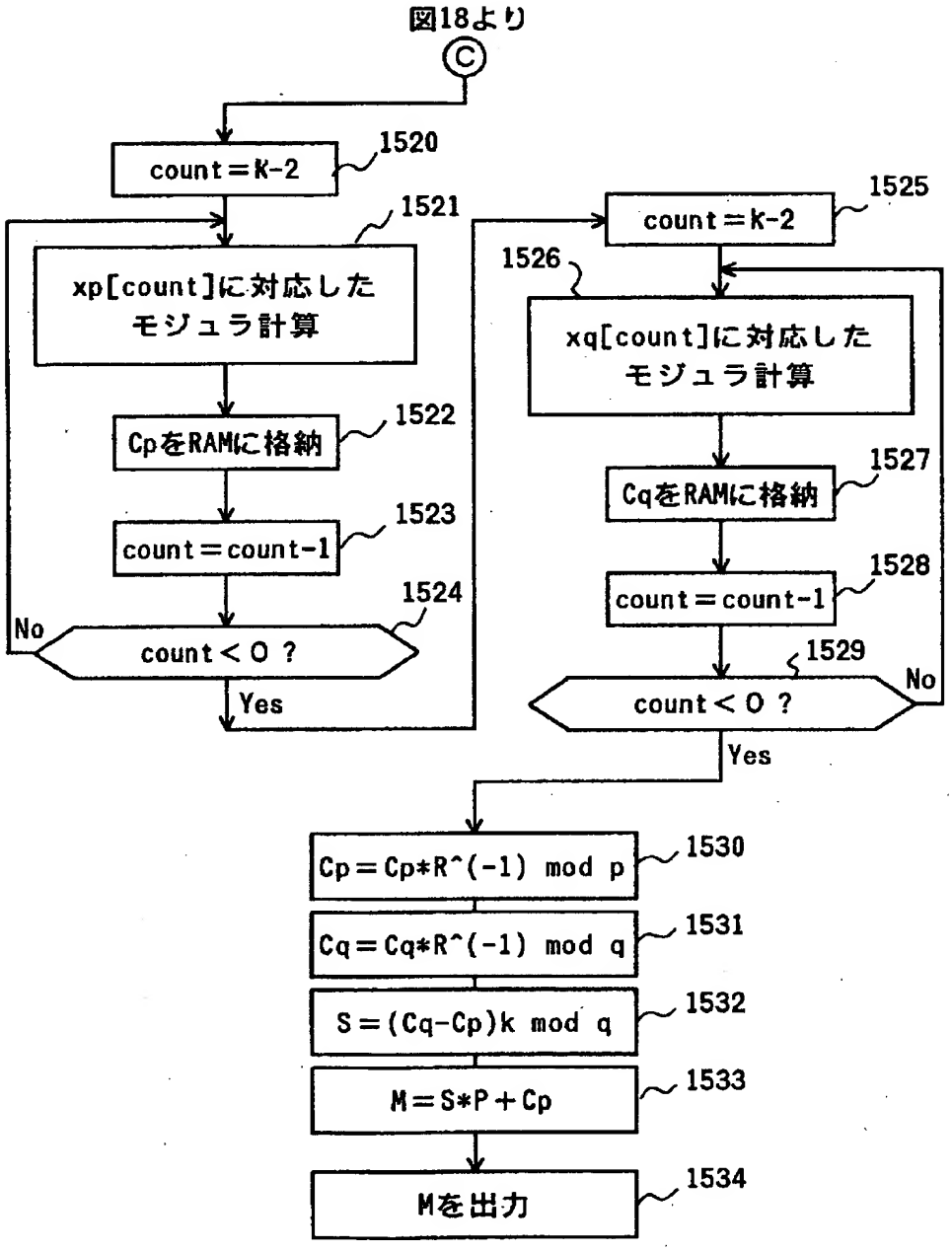
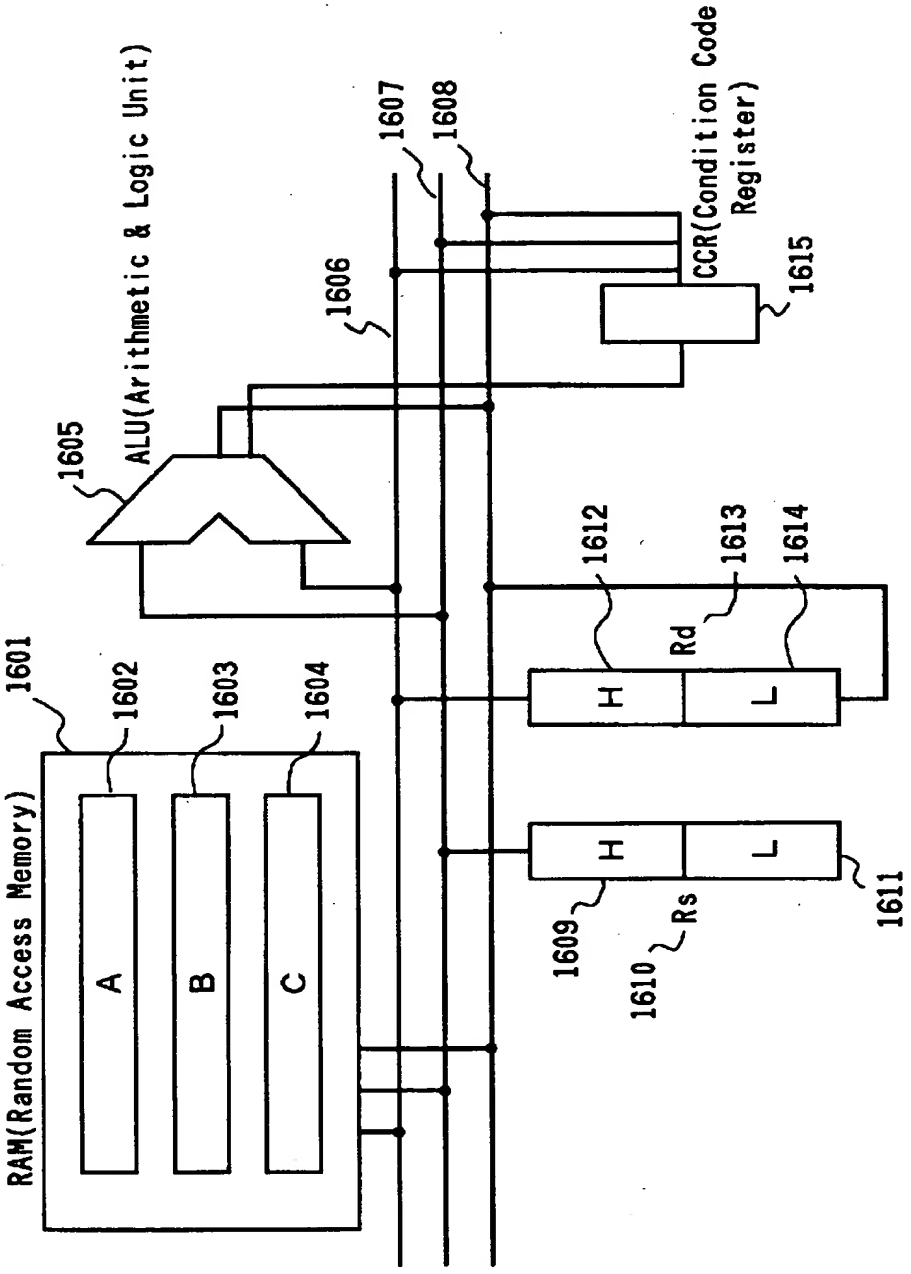


図18より
Ⓒ
count＝K-2
1520
1521
xp[count]に対応した
モジュラ計算
CpをRAMに格納
1522
count＝count-1
1523
1524
No
count＜０？
Yes
count＝k-2
1525
1526
xq[count]に対応した
モジュラ計算
CqをRAMに格納
1527
count＝count-1
1528
1529
No
count＜０？
Yes
Cp＝Cp*R^(-1) mod p
1530
Cq＝Cq*R^(-1) mod q
1531
S＝(Cq-Cp)k mod q
1532
M＝S*P＋Cp
1533
Mを出力
1534

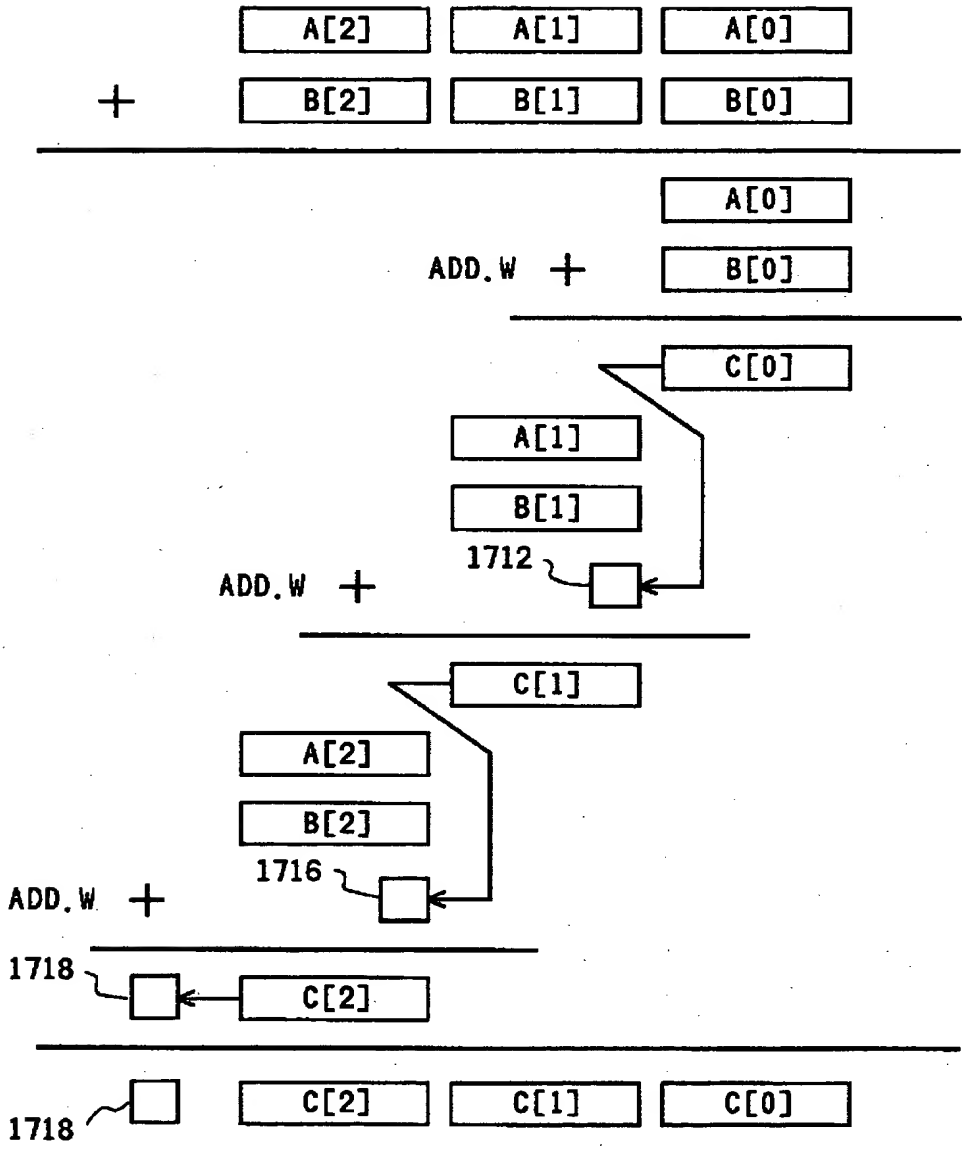
【図２０】

図 20


RAM(Random Access Memory)
A
B
C
1601
1602
1603
1604
ALU(Arithmetic & Logic Unit)
1605
1606
1607
1608
CCR(Condition Code Register)
1615
1612
1613
1614
Rd
H
L
1609
1610
Rs
1611

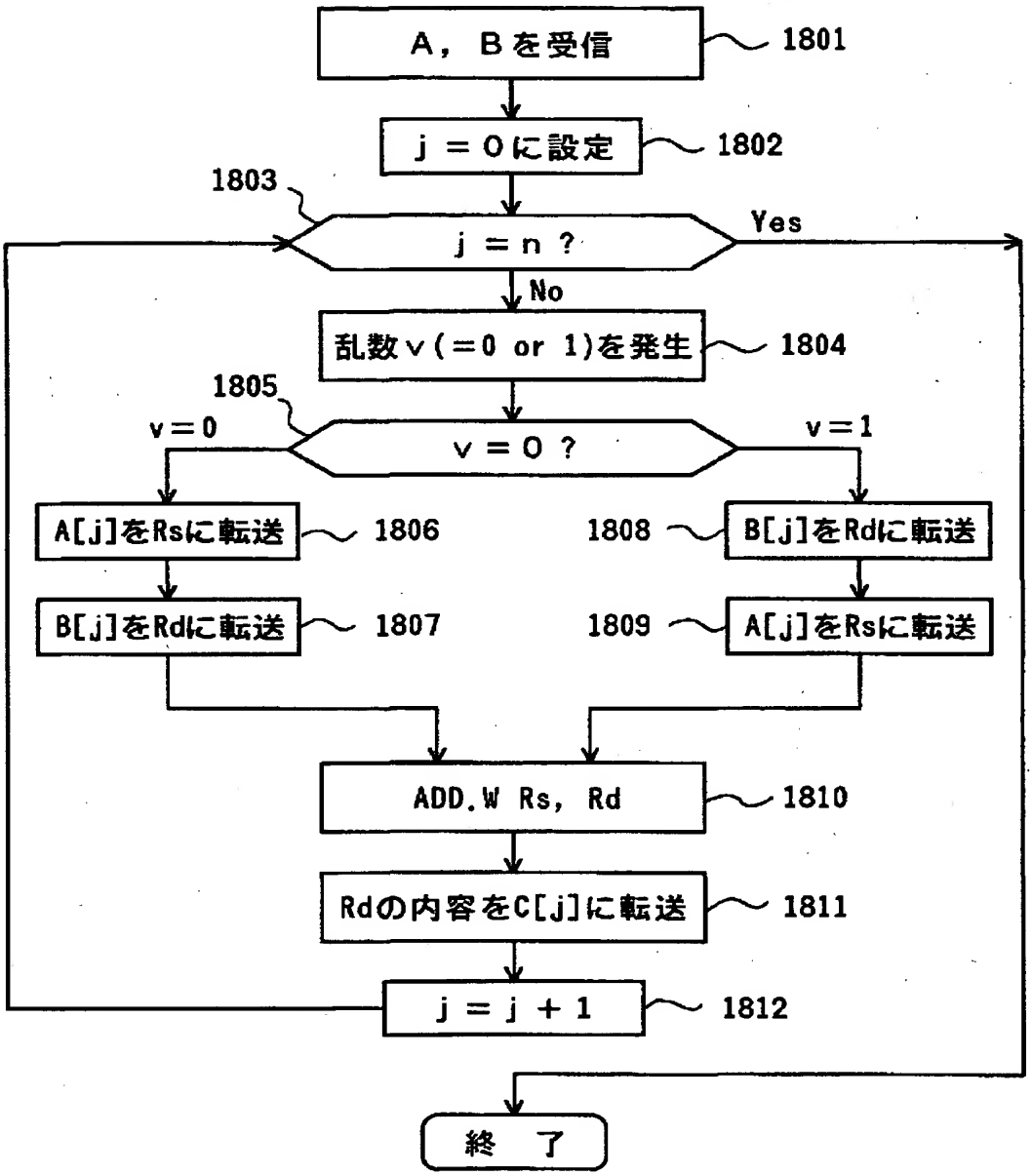
【図２１】

図 21


A[2]
A[1]
A[0]
+
B[2]
B[1]
B[0]
A[0]
ADD.W +
B[0]
C[0]
A[1]
B[1]
1712
ADD.W +
C[1]
A[2]
B[2]
1716
ADD.W +
1718
C[2]
1718
C[2]
C[1]
C[0]

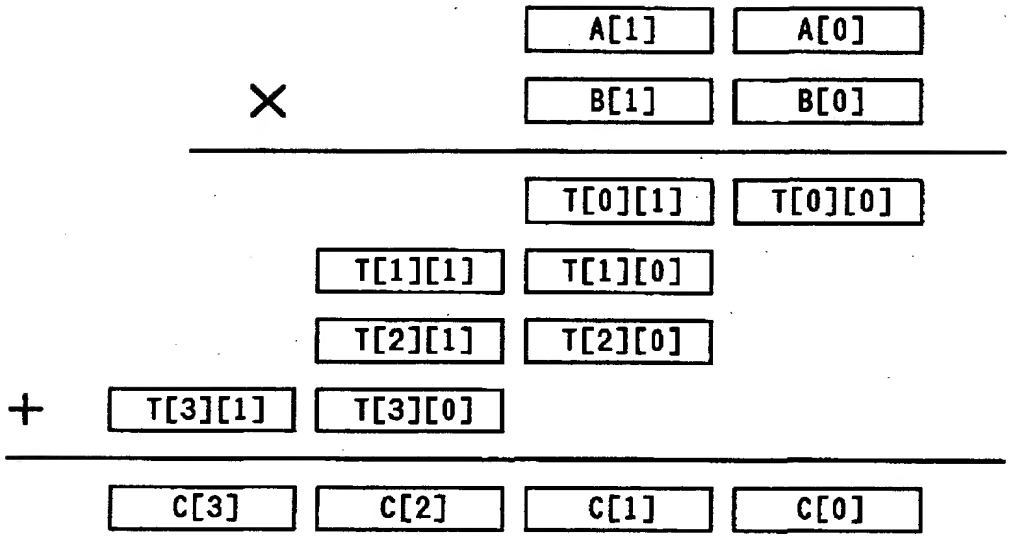
【図２２】

図　22


A，Bを受信
1801
j＝0に設定
1802
1803
j＝n？
Yes
No
乱数v(＝0 or 1)を発生
1804
1805
v＝0
v＝0？
v＝1
A[j]をRsに転送
1806
B[j]をRdに転送
1807
1808
B[j]をRdに転送
1809
A[j]をRsに転送
ADD.W Rs，Rd
1810
Rdの内容をC[j]に転送
1811
j＝j＋1
1812
終　了

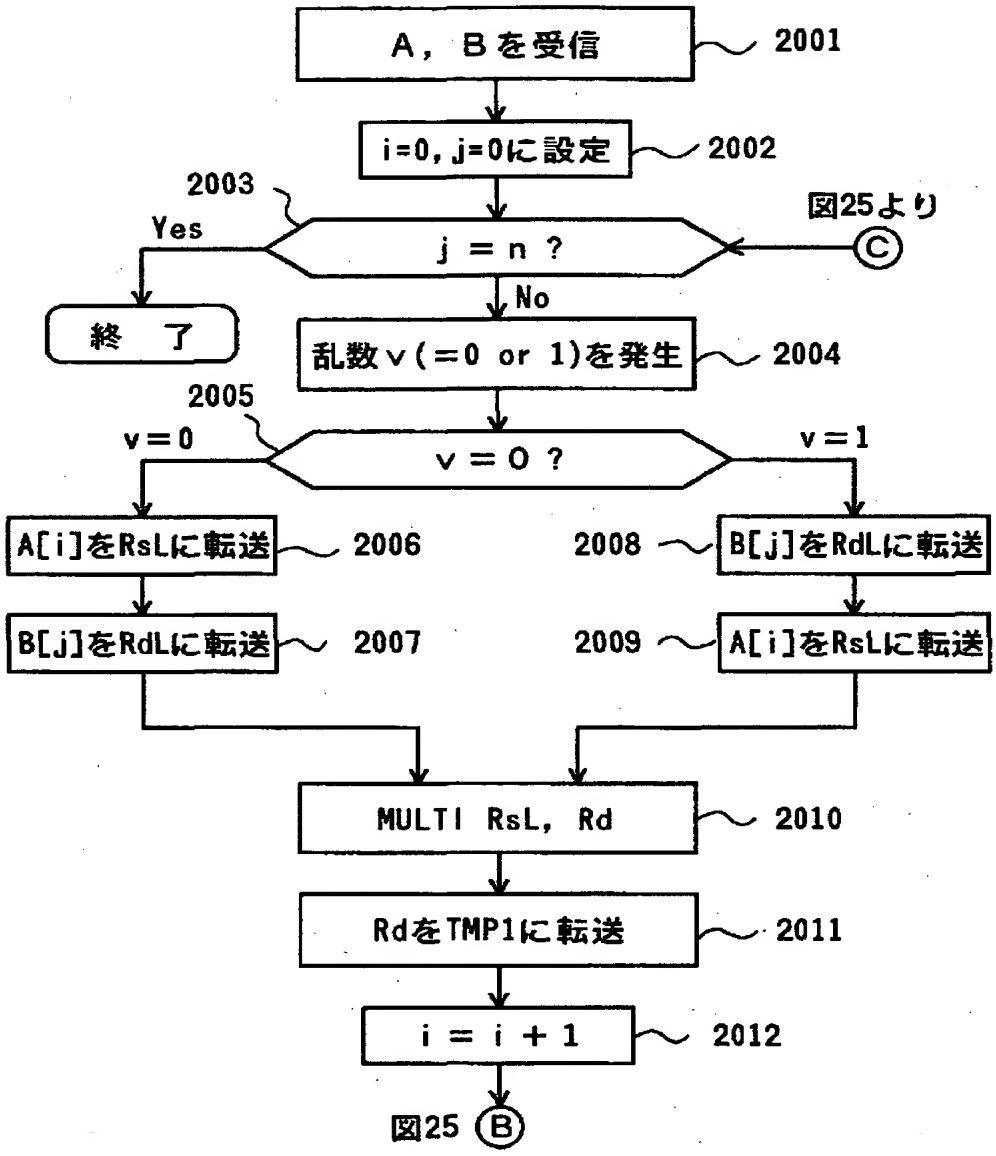
【図２３】

図 23



【図２４】

図　24


A，Bを受信
2001
i=0，j=0に設定
2002
2003
図25より
Yes
j ＝ n ？
C
No
終　了
乱数v(＝0 or 1)を発生
2004
2005
v＝0
v＝0 ？
v＝1
A[i]をRsLに転送
2006
2008
B[j]をRdLに転送
B[j]をRdLに転送
2007
2009
A[i]をRsLに転送
MULTI RsL，Rd
2010
RdをTMP1に転送
2011
i ＝ i ＋ 1
2012
図25
B

【図２５】

図　25

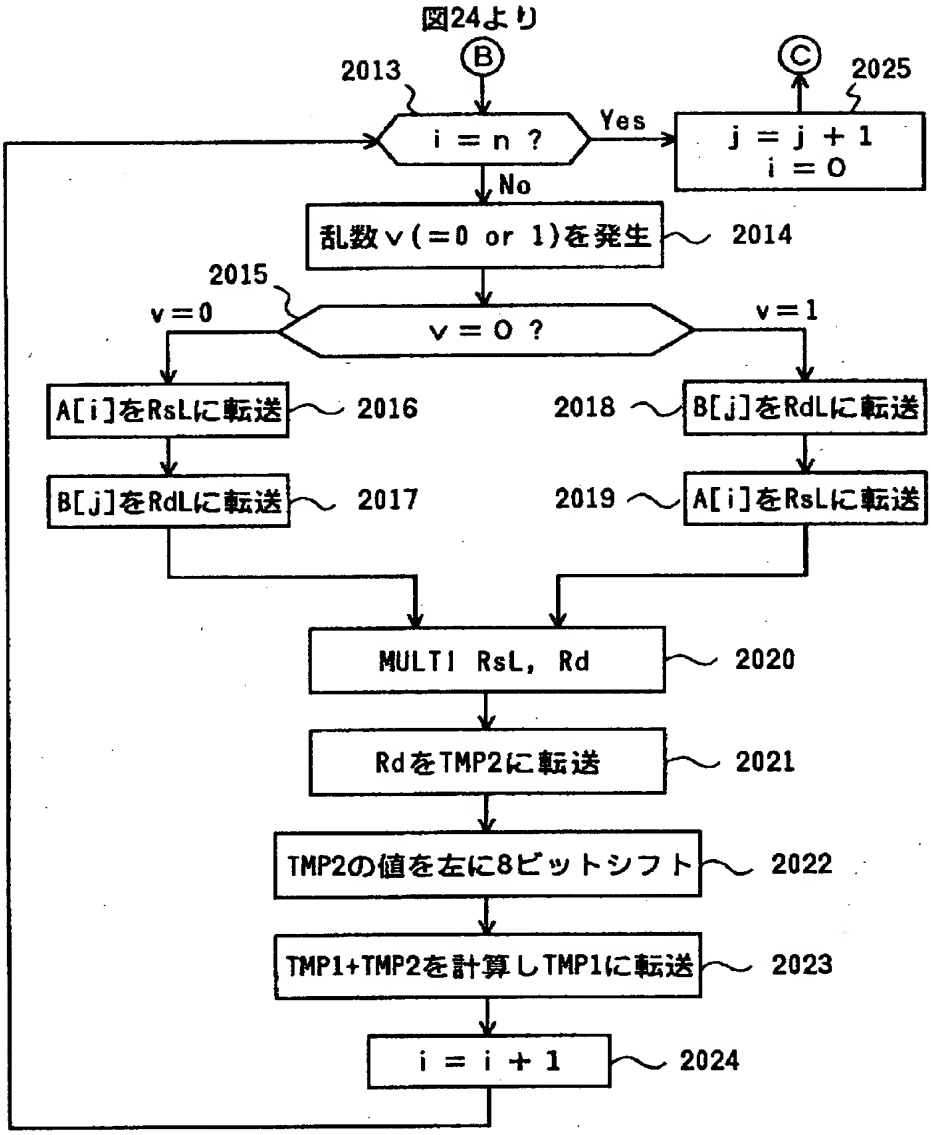
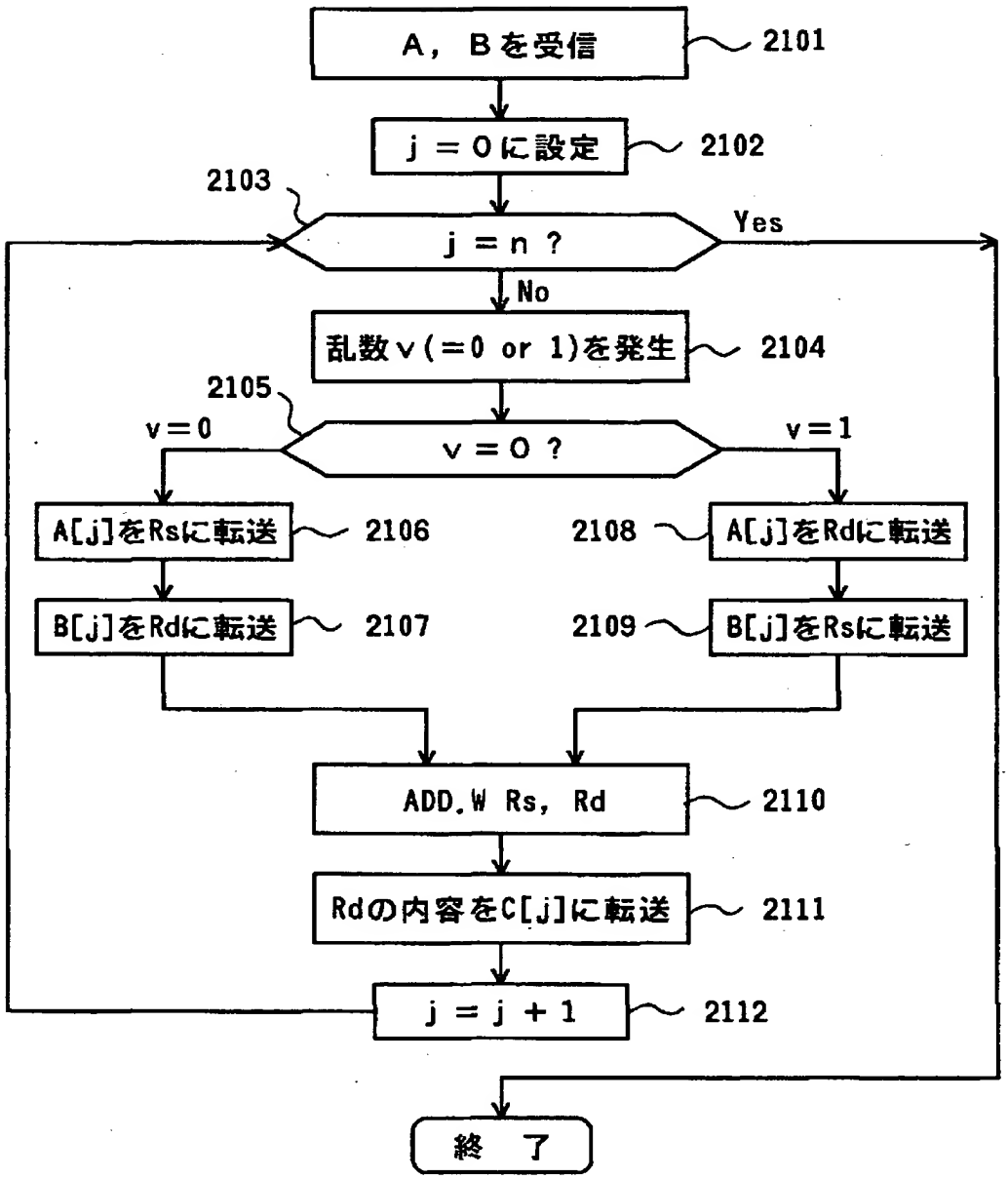


図24より
B
2013
i = n ?
Yes
C
2025
j = j + 1
i = 0
No
乱数v(=0 or 1)を発生
2014
2015
v=0
v = 0 ?
v=1
A[i]をRsLに転送
2016
2018
B[j]をRdLに転送
B[j]をRdLに転送
2017
2019
A[i]をRsLに転送
MULTI RsL, Rd
2020
RdをTMP2に転送
2021
TMP2の値を左に8ビットシフト
2022
TMP1+TMP2を計算しTMP1に転送
2023
i = i + 1
2024

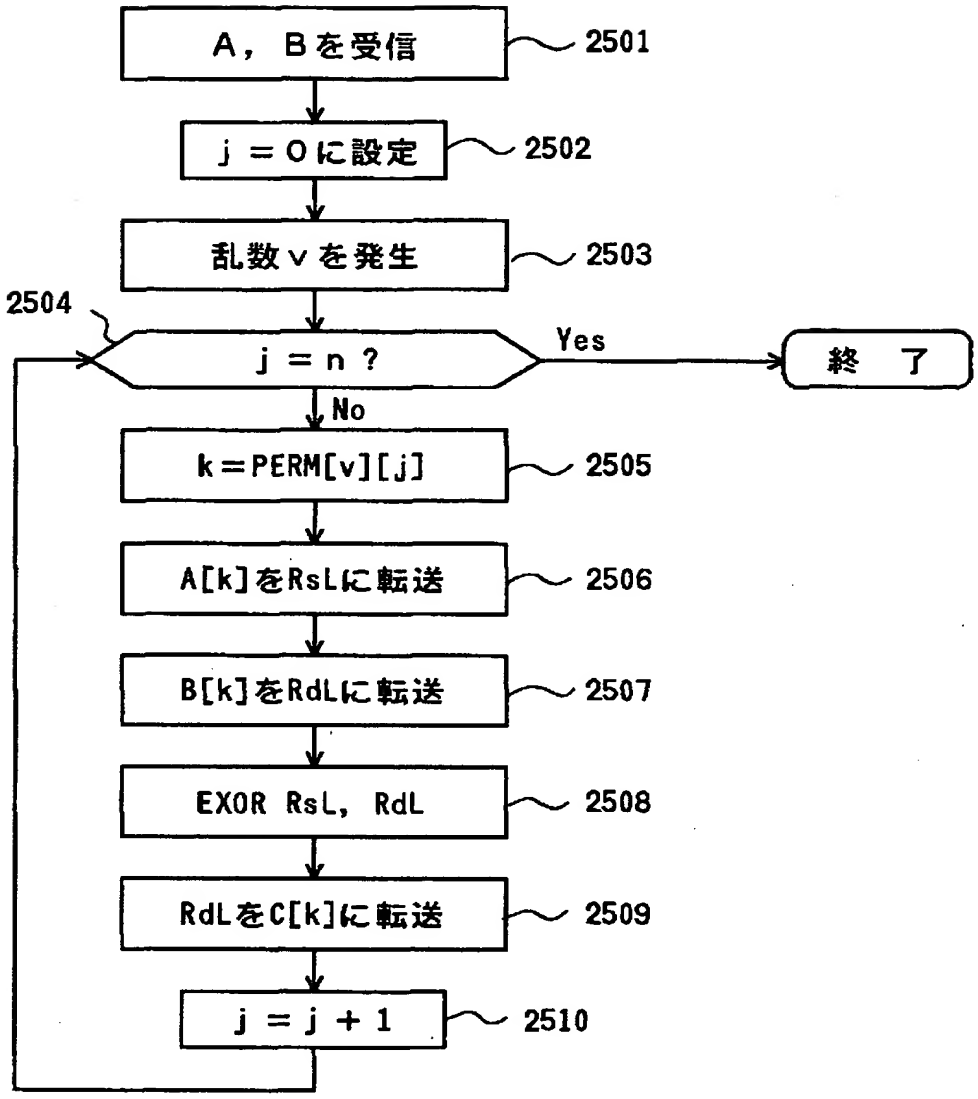
【図２６】

図　26


A，Bを受信
2101
j＝0に設定
2102
2103
j＝n？
Yes
No
乱数v(＝0 or 1)を発生
2104
2105
v＝0
v＝0？
v＝1
A[j]をRsに転送
2106
2108
A[j]をRdに転送
B[j]をRdに転送
2107
2109
B[j]をRsに転送
ADD.W Rs，Rd
2110
Rdの内容をC[j]に転送
2111
j＝j＋1
2112
終　了

【図２７】

図　27


A，Bを受信
2501
j＝0に設定
2502
乱数vを発生
2503
2504
j＝n？
Yes
終　了
No
k＝PERM[v][j]
2505
A[k]をRsLに転送
2506
B[k]をRdLに転送
2507
EXOR RsL，RdL
2508
RdLをC[k]に転送
2509
j＝j＋1
2510

【図２８】

図　28

A	B	A EXOR B
1	1	0
1	0	1
0	1	1
0	0	0

【図２９】

図　29

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

【図３０】

図 30

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

【書類名】　要約書

【要約】

【課題】　ＩＣカードなどに埋め込まれる情報処理装置を利用して暗号処理を行なうに際して、消費電流の波形を観測するタンパーに対抗して、暗号処理と消費電流との関連性を減らす。

【解決手段】　中国人剰余定理に従ってＲＳＡ暗号の復号処理を行なうに際して、ステップ608でxpの単位ビットブロックごとにべき剰余計算を行なって、計算済のビットブロックまでのCpの途中結果をメモリに格納する。ステップ609でxqの単位ビットブロックごとにべき剰余計算を行なって、計算済のビットブロックまでのCqの途中結果をメモリに格納する。ステップ606で乱数を生成し、ステップ607でその乱数の値に応じてステップ608を実行するか、ステップ609を実行するかを決定する。

【選択図】図６

出証特２００１－３０６６０３３

出　願　人　履　歴　情　報

識別番号　　〔０００００５１０８〕

１．変更年月日　　１９９０年　８月３１日
［変更理由］　新規登録
住　所　　東京都千代田区神田駿河台４丁目６番地
氏　名　　株式会社日立製作所